VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

METÓDY PRE ZÍSKAVANIE ASOCIAČNÝCH PRAVIDIEL Z DÁT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN UHLÍŘ

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

METODY PRO ZÍSKÁVÁNÍ ASOCIAČNÍCH PRAVIDEL Z DAT

METHODS FOR MINING ASSOCIATION RULES FROM DATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN UHLÍŘ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VLADIMÍR BARTÍK, Ph.D.

BRNO 2007

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2006/2007

Zadání diplomové práce

Řešitel: **Uhlíř Martin, Bc.**
Obor: Informační systémy
Téma: **Metody pro získávání asociačních pravidel z dat**
Kategorie: Databáze

Pokyny:

1. Seznamte se s problematikou získávání znalostí z dat, přičemž se zaměřte na problematiku získávání asociačních pravidel.
2. Po dohodě s vedoucím vyberte jednu z metod pro získávání asociačních pravidel. Tuto metodu podrobně prostudujte.
3. Navrhněte aplikaci pro implementaci zvolené metody.
4. Vybranou metodu implementujte a ověřte její funkčnost.
5. Zhodnoťte dosažené výsledky a další možné pokračování v tomto projektu.

Literatura:

- Holt, J. D., Chung, S. M.: Efficient Mining of Association Rules In Text Databases, In: Proceedings of CIKM 99, Kansas City, USA, ACM, 1999, pp. 234-242
- Feldman, R., Hirsh, H.: Exploiting Background Information In Knowledge Discovery From Text. Journal of Intelligent Information Systems, Vol. 9, pp. 83-87, Kluwer Academic Publisher, 1997

Při obhajobě semestrální části diplomového projektu je požadováno:

- První tři body zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bartík Vladimír, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 28. února 2006
Datum odevzdání: 22. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Bc. Martin Uhlíř**
Id studenta: 49542
Bytem: Daxnerova 1182/3, 050 01 Revúca
Narozen: 19. 10. 1983, Revúca
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP: Metody pro získávání asociačních pravidel z dat
Vedoucí/školitel VŠKP: Bartík Vladimír, Ing., Ph.D.
Ústav: Ústav informačních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☒ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

Cieľom práce je implementácia metódy Multipass-Apriori pre získavanie asociačných pravidiel z textových dát. Po úvode do problematiky dolovania z dát je spomenutá špecifickosť dolovania znalostí z textových dát. Veľmi dôležitú úlohu v tomto procese zohráva predspracovanie, v tomto prípade najmä použitie stemmingu, a vytvorenie slovníka nepotrebných slov (stopwords). Významu, využitiu a procesu získavania asociačných pravidiel je venovaná ďalšia časť práce. Najväčšia pozornosť je venovaná metóde Multipass-Apriori, ktorá bola naimplementovaná a bol popísaný princíp jej fungovania. Na základe vykonaných testov bol stanovený optimálny spôsob rozdelenia partícií a spôsob usporiadania množín. Pri praktických testoch bola metóda Multipass-Apriori porovnávaná s metódou Apriori.

Kľúčové slová

frekventovaná množina, asociačné pravidlo, Apriori, Multipass-Apriori, stemming, slovník nepotrebných slov, predspracovanie textových dát

Abstract

The aim of this thesis is to implement Multipass-Apriori method for mining association rules from text data. After the introduction to the field of knowledge discovery, the specific aspects of text mining are mentioned. In the mining process, preprocessing is a very important problem, use of stemming and stop words dictionary is necessary in this case. Next part of thesis deals with meaning, usage and generating of association rules. The main part is focused on the description of Multipass-Apriori method, which was implemented. On the ground of executed tests the most optimal way of dividing partitions was set and also the best way of sorting the itemsets. As a part of testing, Multipass-Apriori method was compared with Apriori method.

Keywords

frequent itemset, association rules, Apriori, Multipass-Apriori, stemming, stop words, text data preprocessing,

Citace

Martin Uhlíř: Metódy pre získavanie asociačných pravidiel z dát, diplomová práca, Brno, FIT VUT v Brně, 2007

Metódy pre získavanie asociačných pravidiel z dát

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením Ing. Vladimíra Bartíka, Ph.D.
Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

……
Martin Uhlíř
15.5.2007

Poďakovanie

Chcel by som sa poďakovať Ing. Vladimírovi Bartíkovi, PhD. za poskytnutie odbornej pomoci a usmernenie pri tvorbe diplomovej práce.

Obsah

1 Úvod

Textová podoba je aj v dobe rozširujúcej sa multimedializácie najrozšírenejším spôsobom uchovania informácií. Každý človek na svete v priemere vyprodukuje 333MB dát za rok[1]. Textové dáta sú väčšinou neštruktúrované a ako také nie sú implicitne vhodné na získavanie nových, na prvý pohľad neznámych znalostí z dát. Proces dolovania znalostí, ako sa táto činnosť odborne nazýva bol pôvodne využívaný na získavanie nových znalostí z transakčných databáz, napríklad nákupných košíkov. Textové databáze bývajú omnoho rozsiahlejšie, avšak po vhodnom predspracovaní môže byť podobná činnosť vykonávaná aj nad takýmito dátami.

Jazyk a jeho textová reprezentácia je značne rôznorodá a rozsiahla, preto je prvoradou úlohou predspracovania textových dát najmä znížiť počet položiek (slov) nad ktorými bude dolovanie vykonávané. Štruktúrou jazyka sa zaoberá celá rada vedných odvetví, ako napríklad lexikálna analýza, syntaktická analýza, morfológia atď. Priamym nástrojom na zníženie počtu slov je napríklad využitie slovníka nepotrebných slov a tým odstránenie slov, ktoré majú malú informačnú hodnotu. Ďalším vhodným krokom zníženia počtu tvarov slov je využitie stemmingu, ktorý slúži na redukciu slov na ich slovný základ. Nad takto upravenými slovami už môže byť efektívne vykonané dolovanie. Klasický algoritmus Apriori však pri dolovaní zlyháva najmä pre svoju veľkú pamäťovú náročnosť. Vhodnou náhradou môže byť použitie algoritmu Multipass-Apriori, ktorý rozdeľuje frekventované 1-množiny do partícií a tie postupne spracováva. Takýmto postupom sú získané frekventované k-množiny a z nich následne vygenerované asociačné pravidlá. Tie sú vhodné na reprezentovanie znalosti získaných z textových dát.

Cieľom mojej práce je implementácia algoritmu Multipass-Apriori pre získavanie asociačných pravidiel z textových dát a jeho porovnanie s klasickým algoritmom Apriori. V problematike predspracovania som sa venoval najmä využitiu stemmingu a využitiu slovníka nepotrebných slov pri predspracovaní testových dokumentov kolekcie Reuters-21578.

Diplomová práca naväzuje na semestrálny projekt, v ktorom som naštudoval a spracoval problematiku získavania znalostí z textových dát, problematiku predspracovania a využitia rôznych metód pre získavanie asociačných pravidiel z dát.

1.1 Prehľad kapitol

Po úvodnej kapitole nasleduje kapitola, v ktorej sú vysvetlené pojmy z oblasti získavania znalostí z dát. Takisto sú tu uvedené kroky v ktorých získavanie znalostí prebieha. Podstatná časť tejto kapitoly sa venuje špecifickému typu dolovania z dát, a to text miningu, teda získavaniu znalostí z textových dát. Sú tu popísané jednotlivé fázy dolovania, úlohy text miningu ako aj oblasti jeho využitia.

Rozsiahlou kapitolou je kapitola tretia venovaná predspracovaniu. Po všeobecných dôvodoch k predspracovaniu nasleduje podkapitola zaoberajúca sa predspracovaním textových dát. Po uvedení špecifík textových dát sú podrobnejšie rozobrané techniky predspracovania textových dát, ako napríklad lematizácia, syntaktická analýza, morfologická disambiguácia, atď. V ďalšej časti je uvedený spôsob predspracovania textových dát využitý v aplikácia, a to proces tokenizácie, syntaktickej analýzy využitia Porterovho stemmovacieho algoritmu a slovníka nepotrebných slov (stop words).

Kapitola štvrtá sa zaoberá asociačnými pravidlami, ich definíciou, tvorbou, využitím a spôsobom získavana z textov. V závere kapitoly sú uvedené príklady asociačných pravidiel získaných z niektorých dokumentov kolekcie dát Reuters.

V piatej kapitole je pomocou pseudokódu opísaný základný algoritmus na získavanie asociačných pravidiel z dát, a to algoritmus Apriori. V ďalšej časti je popísaný princíp fungovania algoritmu Multipass-Apriori.

V šiestej kapitole dochádza k porovnaniu oboch algoritmov z hľadiska pamäťovej a časovej efektivity. Na konkrétnom prípade je podrobne vysvetlený postup vytvárania frekventovaných množín pomocou algoritmu Multipass-Apriori. Významným faktorom ovplyvňujúcim pamäťovú náročnosť algoritmu Multipass-Apriori je voľba veľkosti partície. Tejto problematike je venovaný zvyšok tejto kapitoly.

Aplikácii ako takej sa venuje kapitola siedma. Je v nej uvedený prehľad tried spolu s ich popisom. Ďalej nasleduje popis štruktúry konfiguračného súboru, ktorým je aplikácia riadená. V závere kapitoly je ukážka vzhľadu a výstupu aplikácie.

Šiesta kapitola je záver, kde sú zhrnuté výsledky práce a možné smery ďalšieho rozvoja.

2 Základné pojmy

2.1 Získavanie znalostí z databáz a dolovanie z dát

Prudký rozmach informačných technológií so sebou prináša nahromadenie obrovských objemov dát. Tieto dáta môžu v sebe obsahovať množstvo veľmi cenných vedecky, komerčne, obchodne i inak zužitkovateľných vzťahov a závislostí, ktoré dovtedy neboli známe alebo podložené. Pojmy ako získavanie znalostí z databáz (Knowledge Discovery in Databases - KDD) a dolovanie dát (Data Mining) sa v tejto súvislosti v posledných rokoch veľmi často objavujú v informatickom priemysle najmä kvôli ich širokému spektru použitia v podobe získavania užitočných informácií a znalostí z týchto dát.

Získavanie znalostí z databáz sa dá charakterizovať ako netriviálne získavanie implicitných, pred tým neznámych a potenciálne užitočných informácií z dát [1].

Dolovanie dát znamená získavanie alebo „dolovanie“ znalostí z veľkých objemov dát[2].

Na prvý pohľad by sa mohlo zdať, že oba pojmy sú totožné. V ďalšom texte ich skutočne budeme brať ako synonymá, avšak prísne vzato dolovanie z dát je len jedným z krokov získavania znalostí z databáz.

Získavanie znalostí z databáz KDD je interaktívny a iteratívny proces tvorený krokmi:

1. **Selekcia (Selection):** vytvorenie cieľovej množiny dát - z veľkého množstva dát vyberieme tú množinu, ktorá môže obsahovať užitočné dáta alebo ešte predtým zlúčime viac heterogénnych dátových zdrojov do jedného. Môžeme sa sústrediť aj na konkrétnu množinu atribútov.
2. **Predspracovanie (Preprocessing):** vyčistenie vzoriek dát, ktoré sme vybrali vo fáze selekcie od nesprávnych, chýbajúcich, nerelevantných dát a od šumu. Ďalej doplnenie chýbajúcich hodnôt atribútov, vynechanie veľmi odľahlých hodnôt a použitie vhodných algoritmov pre predspracovanie dát (napr. výpočet nových, užitočnejších atribútov z existujúcich).
3. **Transformácia (Transformation):** formát výstupu systému pre predspracovanie dát, ktorý sme použili v predchádzajúcej fáze je potrebné transformovať do formátu, ktorý je vhodný pre systém samotného Data Mining-u. Ďalším cieľom transformácie je aj redukcia dát, ktorá umožní skúmanie len jednej vzorky z databázy.
4. **Dolovanie (Data mining):** reprezentuje kľúčovú časť celého KDD procesu. V časti Data Mining je aplikovaný na transformované dáta nejaký učiaci sa, analyzačný, objavovací algoritmus - sumarizácia, klasifikácia, regresia, zhlukovanie atď. s účelom získavania

generalizácie dát. Bez dobrej prípravy dát sa nedajú dosiahnuť dobré výsledky, akokoľvek rozvinutý je dolovací algoritmus.

5. **Evaluácia/interpretácia (Evaluation/interpretation):** v tejto fáze sú interpretované nájdené vzory.

Dolovanie dát môže byť v princípe aplikovateľné na ľubovolný súbor informácií (dát). Z najznámejších spomeňme relačné databáze, dátové sklady, transakčné databáze, pokročilé databázové systémy zahŕňajúce objektovo-orientované a objektovo-relačné databáze, špecifické aplikačne orientované databáze ako napríklad priestorové databáze, multimediálne databáze a textové databáze.

2.2 Získavanie znalostí z textu - text mining

Analogicky k data mining-u, ktorý extrahuje užitočné informácie z veľkého množstva dát hocijakého typu, je text mining procedúra aplikovaná na textové dáta účelom nájsť zaujímavé pravidelnosti, sémantické a abstraktné informácie. Dôvod oddelenia týchto dvoch oblastí spočíva v rozdielnosti dát, ktoré spracúvavajú. Hlavný rozdiel je v tom, že kým v klasickom data mining-u pracujeme so štruktúrovanými dátami, v text mining-u sú vzory extrahované zo zdrojov v prirodzenom jazyku. Táto rozdielnosť viedla k potrebe vytvoriť nové metódy predspracovania textu.

Text je vo všeobecnosti hodne neštruktúrovaný - má len malú alebo takmer žiadnu vnútornú štruktúru, je amorfný a ťažko sa s ním pojednáva. Keďže textový formát je veľmi flexibilným spôsobom popisu a uchovania rozličných typov dát, uskladňujú obrovské množstvo informácií ako text, ktoré potencionálne môžu obsahovať veľké množstvo poznatkov. Tieto informácie sú však zakódované formou, ktorú sa dá len ťažko dešifrovať automaticky. V dnešnej dobe je väčšina (takmer 80%) všetkých informácií uložená v podobe textových dokumentov [4].

Ako jednotlivé fázy KDD procesu, aj text mining môže vyzerať nasledujúcim spôsobom [7]:

1. Selekcia - výber dát: vyhľadávanie, kategorizácia a zhlukovanie dokumentov.
2. Predspracovanie dát - lematizácia, morfologická desambiguácia, parciálna syntaktická analýza.
3. Transformácia - konštrukcia atribútov, zhlukovanie termov.
4. Dolovanie.
5. Interpretácia - sumarizácia a vizualizácia.

Dolovanie znalostí z textu sa vo veľmi veľkej miere využíva v oblasti filtrovania a vyhľadávania (napr. spamové filtre). Uplatnenie text mining-u nájdeme aj v nasledujúcich oblastiach:

- Marketing - zhromažďovanie štatistík, prieskum trhu.
- Manažérske rozhodovanie - predpovedanie trendu.
- Klasifikácia a organizácia dokumentov podľa ich obsahu.

- Identifikovanie väzieb medzi dokumentmi (napr. genetika - vzájomné pôsobenie proteínov s inými proteínmi na základe hľadania slov, ktoré sa spoluobjavujú v článkoch, ktoré sa venujú predpovedaniu takýchto vzájomných interakcií; biológia - hľadanie možných príčin zriedkavých chorôb metódou nepriamych spojení v rozdielnych prameňoch biologickej vedeckej literatúry)
- Získavanie asociačných pravidiel z dokumentov.

Zdrojmi dát pri dolovaní môžu byť napríklad elektronická pošta, internetové dokumenty (poznámky, prezentácie), technické správy, ktoré popisujú nové technológie, internetové noviny, novinové archívy, www stránky atď. Veľká časť textových dát je v súčasnosti prístupná na internete, avšak aj vo firemnej sfére existuje veľké množstvo elektronických dát, nakoľko vďaka informatizácii firmy uchovávajú svoje dáta v elektronickej podobe.

Na dolovanie v textoch sa môžeme aj pozrie ako na činnosť skladajúcu sa z dvoch častí [4]:

1. úprava textu (text refining) - text je prevedený pomocou metód predspracovania do tzv. medzipodoby.
2. získavanie znalostí (knowledge destilation) - v tejto časti spracovania sú odvodzované vzory alebo znalosti z medzipodoby.

2.2.1 Úlohy text mining-u

Hlavné úlohy text mining-u sú nasledujúce [4]:

- *Kategorizácia textov (Text categorization*) - pri úlohe kategorizácie textov sú dokumenty automaticky zaraďované do jednotlivých preddefinovaných tried. Môžu byť delené podľa obsahu, žánru, autora, atď.
- *Extrakcia informácií (Information extraction)* - prevádza sa neštruktúrovaný text do štruktúrovaného formátu (napríklad do databázy) podľa obsahu, aby tak dodal informácie ďalším metódam v procese spracovania. Prístup úlohy je vybraný podľa štruktúry textu. Tá môže nadobúdať tieto podoby:
    - neštruktúrovaný text (free text) - text celými gramatickými vetami, ktoré umožňujú spracovanie pomocou metód spracovania prirodzeného jazyka (NLP – viď 3.2.2). Vzory sú získané pomocou syntaktickej a sémantickej analýzy;
    - štruktúrovaný text - text s dopredu definovanou štruktúrou, v ktorom sú vzory oddelené určitým pevne daným spôsobom;
    - semi-štruktúrovaný text - text, ktorý nie je gramaticky štruktúrovaný, ale má jednoduchú štruktúru napr.: text písaný heslovitým spôsobom.
- *Zhlukovanie textov (Clustering)*: Zhlukovanie textov je plne automatický proces, ktorý rozdelí dokumenty do skupín podľa nejakých kritérií. Dokumenty sú často rozdelené podľa obsahu. K identifikácii témy dokumentov zhlukovacie nástroje používajú slová, ktoré sú

bežné v dokumentoch danej skupiny. Cieľom tejto analýzy je rozlíšiť množinu skupín (cluster), v ktorých je podobnosť s inými množinami skupín minimálna, ale vnútorná podobnosť dokumentu je maximálna.

- *Filtrovanie textov*: Tento typ úlohy dolovania v textoch slúži k vyhľadávaniu informácií v dynamickom dátovom prúde na základe konkrétnych podmienok.

Ďalšími možnosťami dolovania v textoch sú:

- automatická identifikácia jazyka dokumentov,
- automatické rozdelenie dokumentov,
- určenie autora dokumentov,
- rozpoznávanie plagiátorstva,
- automatický preklad textov a iné.

3 Predspracovanie

3.1 Dôvody k predspracovaniu

Jeden z dôvodov spočíva v rozličnosti zdrojov dát. Môžu pochádzať od ľudí, od senzorov a ani jeden z nich nie je 100 % spoľahlivý zdroj. Preto môžeme mať veľa problémov s týmito dátami. Než začneme aplikovať algoritmy dolovania z dát (DM algoritmy), je potrebné ich vyradiť. Keď neeliminujeme chyby, DM algoritmy nám môžu dať úplne iné výsledky, než v prípade predspracovaných dát. Rastúce množstvo dát, ktoré je hlavne dôsledkom moderného „process monitoring“-u a systému zbierania dát, vyžaduje používanie efektívnych techník predspracovania [8].

Ďalším dôvodom predspracovania dát môže byť napríklad, že dáta nie sú v takej podobe, akú vyžaduje systém, ktorým v nich chceme dolovať. Preto pred samotným dolovaním ich musíme určitým spôsobom upraviť.

S dátami reálneho sveta je vždy nejaký problém. Charakter problémov je taký, že sa vyskytujú aj napriek ľudskej snahe ich eliminovať. Problémy s dátami môžeme členiť do troch hlavných skupín[8]:

1. *príliš veľa dát* - k tejto skupine patria problémy, keď disponujeme s veľkým množstvom dát, ktoré nám sťažujú dolovanie užitočných informácií.
    - Narušené dáta
    - Extrakcia čŕt
    - Nepodstatné dáta
    - Obrovský objem dát
    - Numerické / symbolické dáta
2. *príliš málo dát* - tu sa vyskytuje presne opačný stav, než v prvom prípade. Vyhľadávanie informácií nám sťažuje situácia, že máme k dispozícii málo dát.
    - Chýbajúce atribúty
    - Chýbajúce hodnoty atribútov
    - Malé množstvo dát
3. *poškodené dáta* - v tejto skupine sa problém nachádza v tom, že máme dáta z viacerých zdrojov alebo ich získame vo viacerých formátoch.
    - Nekompatibilné dáta
    - Viac dátových zdrojov
    - Rozličná úroveň charakteru dát

3.2 Predspracovanie textových dát

3.2.1 Špecifiká textových dát

Metódy dolovania v textoch pracujú s dátami, ktoré majú kvôli svojej štruktúre odlišné vlastnosti než dáta používané v data miningu. Metódy musia veľmi často zápasiť s veľkými súbormi dát, častými zmenami dát v nich uložených, ich veľkým šumom a inými problémami. Najväčším problémom však je, že texty nie sú navrhované pre jednoduché a pohodlné využívanie počítačmi, ale pre ľudí. Texty majú zle definovanú štruktúru, zložitú sémantiku, často sa môžu objavovať dvojzmyselnosti a sú písané v mnohých jazykoch. Výsledkom toho všetkého je úplne odlišný prístup dolovania v textoch než dolovanie v ostatných dátach[10].

3.2.2 Spracovanie prirodzeného jazyka

Spracovanie prirodzeného jazyka (Natural language processing - NLP) je efektívnym nástrojom pre text mining. Táto disciplína sa zaoberá budovaním jazykových zdrojov a nástrojov, ktoré umožňujú uchopiť prirodzený jazyk strojovými prostriedkami. Text prirodzeného jazyka je možné analyzovať v štyroch rovinách, ktoré na seba postupne nadväzujú.

Sú to roviny [9]:

1. morfologická
2. syntaktická
3. sémantická
4. pragmatická

Hlavnými spracovávanými jednotkami na morfologickej úrovni, z ktorých sa skladajú slová, sú morfémy. Rovina syntaktická skúma stavbu vety, t.j. ako sú budované jednotlivé druhy výrazov z jednoduchších jednotiek. Sémantická rovina sa snaží zachytiť význam vety v závislosti na jej syntaktickej štruktúre - hľadá vzájomné súvislosti medzi významom a štruktúrou. Na štvrtej úrovni, ktorá sa nazýva niekedy aj textová rovina, kde základnou spracovávanou jednotkou je text, sú hľadané vzájomné vzťahy medzi významom jednotlivých viet.

Pri predspracovaní textových dát sú používané nasledujúce techniky:

- Lematizácia
- Parciálna syntaktická analýza
- Morfologická desambiguácia
- Výber atribútov

3.2.3 Lematizácia

Lematizácia znamená priradiť k slovu jeho základný tvar (tzv. lemma). Toto sa väčšinou deje na morfologickej úrovni a vykonávajú ju morfologické analyzátory. Tie sú ešte schopné určiť všetky prípustné gramatické interpretácie ako slovný druh a hodnoty všetkých gramatických kategórií, ktoré sa s tvarom slova spájajú (napr. pád, číslo a rod). Gramatický význam je označovaný gramatickou značkou, čo je reťazec gramatických kategórií s ich príslušnými hodnotami.

3.2.4 Syntaktická analýza

Na syntaktickej rovine je úlohou nájsť a formulovať pravidlá, podľa ktorých sa dajú kombinovať slová do skupín a skupiny do vetných schém [9]. Skupiny (zložky) sú reťazce slov spájajúce sa k sebe podľa riadiaceho slova, ktoré nemožno vynechať, t.j. celá skupina sa dá redukovať práve na toto slovo.

Syntaktická analýza sa zaoberá určením závislostnej alebo zložkovej štruktúry vety. Závislostná štruktúra je založená na vzťahu syntaktickej závislosti, čo je vzťah medzi riadiacim vetným prvkom a prvkom jemu podriadeným. Analyzovať štruktúru vety z tohto hľadiska znamená nájsť vo vete všetky skladbové dvojice, t.j. dvojice (riadiaci prvok, závislý prvok) [9].

3.2.4.1 Problémy syntaktickej analýzy

Hlavným problémom pri syntaktickej analýze je nejednoznačnosť, ktorými sa prirodzené jazyky vyznačujú. Nejednoznačnosti sa vyskytujú na lexikálnej a štruktúrnej úrovni. Štruktúrnu nejednoznačnosť je možné ukázať napríklad na vete „Vypočúval ho s fajkou v ústach.“, kde nie je jasné, či fajku mal v ústach ten, kto vypočúval alebo ten, koho vypočúvali.

3.2.4.2 Parciálna syntaktická analýza

Cieľom štandardných syntaktických analyzátorov je previesť pokiaľ možno úplnú a presnú analýzu vstupnej vety. Predpokladajú, že gramatika pokrýva celý spracovávaný jazyk a potom analyzátor vyhľadá najlepšiu analýzu [9]. Taký prístup ale nie je možné použiť pre texty obsahujúce chyby. Texty, ktoré sú určené k automatickému spracovania však nikdy nebývajú bezchybné. Riešením na tento problém je parciálna analýza. Tá sa snaží získať syntaktické informácie z potencionálne chybného textu na úkor úplnosti a hĺbky analýzy. Hlavnou myšlienkou parciálnej analýzy je nájdenie vetných skupín, ktorých rozpoznanie vyžaduje minimálne syntaktické znalosti a stačí k nim pomerne jednoduchá gramatika [9].

3.2.5 Morfologická desambiguácia

Morfologická analýza značne uľahčuje analýzu na vyšších úrovniach. Problémom ale je, že neberie do úvahy textový kontext analyzovaného slova, dokonca i viacslovné pojmy sú vyhodnocované

jednotlivo. Keďže čeština (ako aj slovenčina) je syntetický jazyk, používa k vyjadreniu určitého jazykového javu spravidla vhodný tvar slova na rozdiel od jazykov analytických, ktoré k rovnakému účelu využívajú skôr pomocné slová. Takým jazykom je napríklad angličtina. Výsledkom tohto faktu je, že množstvo slov má viac možných interpretácií a tak morfologická analýza poskytuje nejednoznačný výsledok: niekoľko možných gramatických interpretácií, ale niekedy aj niekoľko možných základných tvarov. Príklad nejednoznačnosti na úrovni lemmatu nájdeme vo vetách: „On je obed.“ a „On je študent.“. Kým základný tvar slova *„je“* je v prvej vete *jesť*, v druhej vete je *byť*. Na úrovni značiek môžeme spomenúť tvar slova kosť - *„kosti“* - môže byť v druhom, v treťom aj v šiestom páde jednotného čísla[10].

Určiť správnu gramatickú interpretáciu slova v danom kontextu pre počítače je veľmi zložitý problém. Morfologická desambiguácia sa zaoberá práve s týmto problémom.

3.2.6 Výber atribútov

Výber atribútov hrá veľkú rolu aj pri predspracovaní textových dát. V tejto oblasti máme zdrojové dáta k dispozícii v obrovskom množstve. Pri predspracovaní často potrebujeme selektovať dáta, čo je nevyhnutné pre naše konkrétne dolovanie a čo nie. Tým pádom môžeme urýchliť proces text mining-u. Do tejto skupiny predspracovania patria napríklad extrakcia informácií alebo filtrovanie textov.

3.2.7 Korpusy

V počítačovej lingvistike pod pojmom korpus chápeme rozsiahly, vnútorne štruktúrovaný a ucelený súbor písaných textov prirodzeného jazyka, ktoré sú elektronicky uložené a spracovateľné [9]. Korpusy môžu byť všeobecné alebo špecializované, takisto aj značkované alebo neznačkované. V korpusoch sa nachádzajú jazykové dáta v prirodzenej podobe, teda obsahujú aj veľa chýb. S nástrojmi, tzv. korpusovými manažérmi, je možné ich skúmať a z nich vyvodzovať rôzne teoretické informácie. Zďaleka v širšom spektre používajú korpusy obsahujúce okrem čistých textov aj ďalšie jazykové informácie (značky na úrovni slov alebo na úrovni viet).

3.2.8 Stemming

Stemming je proces redukovania skloňovaných, časovaných alebo odvodených slov na ich koreň alebo slovný základ - vo všeobecnosti pre slová v písanej podobe. Stem (slovný základ) nemusím byť identický s morfologickým koreňom slova, obyčajne je dostačujúce, keď súvisiace slová sú namapované na rovnaký základ, aj keď tento nie je právoplatným koreňom. Problematika stemmingu je dlhodobým problémom v počítačovej vede, prvý dokument na téma stemming bol publikovaný v roku 1968. Proces stemmingu, je hojne využívaný vo vyhľadávacích algoritmoch, rozširovaní dotazov, indexovaní a ostatných problémoch spracovania prirodzeného jazyka. Stemmovacie programy sa niekedy nazývajú aj ako stemmovacie algoritmy alebo stemmery.

Algoritmus stemmingu pre angličtinu by mal identifikovať reťazec „fishing“ ako aj „fished“, „fish“, „fisher“ ako slovo vzniknuté zo základu „fish“ a napríklad slovo „stemmer“, „stemming“, „stemmed“ ako slová odvodené od základu „stem“[12].

Existuje niekoľko stemmovacích algoritmov. Jednoduchým je napríklad S-stemmer v ktorom je odstránených iba niekoľko bežných ukončení slov: „ies“, „es“ „s“ (s niekoľkými výnimkami). Jeho výsledky sú však konzervatívne a zriedka produkuje výsledky, ktoré prekvapia užívateľa.

Ďalšou stemmovaciou metódou je Levinov stemmer využívajúci najdlhší algoritmus zhôd a zoznam výnimiek na odstránenie viac než 260 rozličných prípon. Algoritmus sa vyznačuje značnou agresivitou pri odstraňovaní prípon[13].

Stemmer, ktorý sa stal de-facto štandardným algoritmom na stemming anglického jazyka bol napísaný Martinom Porterom a bol publikovaný v júli 1980 v čísle časopisu Program. Pri porovnávaní tohto algoritmu s ostatnými uvedenými vykazoval Porterov stemer najlepšie výsledky čo sa týka úspešnosti (97%) a 90% pokrytie čo sa týka spojení. Preto je práve Porterov stemmovací algoritmus využívaný v aplikácii[13].

3.3 Predspracovanie použité v aplikácii

Na to, aby mohlo byť z textových dokumentov získané asociačné pravidlá, potrebujeme dokumenty upraviť do vhodnej podoby. Proces predspracovania v aplikácii je zameraný na kolekciu dát Reuters-21578. V princípe je nutné vytvoriť z textového dokumentu súbor obsahujúci transakčnú databázu, kde každý riadok je uvedený znakom „T“ nasledovaným číslom transakcie (napr. T254). Za týmto označením už nasledujú slová z dokumentu, ktoré prešli procesom predspracovania. Nad týmto výstupným dokumentom bude vykonávaný proces získavania frekventovaných množín a neskôr aj asociačných pravidiel. Výstupom je aj súbor v ktorom sú za rovnakým označením transakcie (napr. T254) názvy titulkov príslušných dokumentov.

3.3.1 Kolekcia dát Reuters-21578

Ako už názov napovedá, jedná sa o novinové články agentúry Reuters uverejnené v roku 1987. Kolekcia pozostáva z 22 súborov, pričom v každom z nich sa nachádza 1000 dokumentov v SGML formáte. Podrobný formát súborov je zachytený v SGML DTD súbore a vysvetlený v README súbore, pričom oba sú súčasťou tar.gz balíčka spolu s dokumentmi[14]. Z množstva značiek použitých v každom dokumente sú pre naše účely potrebné a použité iba značky:

- `<REUTERS...>` ktorá určuje začiatok a značka `</REUTERS>` ktorá určuje koniec dokumentu
- `<TITLE>` a `</TITLE>` vymedzujúce titulok dokumentu
- `<BODY>` a `</BODY>` vymedzujúce textový obsah dokumentu

- <TEXT TYPE="BRIEF" ktorá značí, že tento dokument je stručný

Všetky ostatné značky sú ignorované, pretože ich význam pre účely získavania asociačných pravidiel v tomto prípade nie je dôležitý.

3.3.2 Tokenizácia

Na najnižšej úrovni sú dokumenty reprezentované ako postupnosť znakov. Dokument je čítaný po riadkoch a tie sú postupne rozdelené na jednotlivé lexikálne jednotky - tokeny (preto sa tento proces nazýva tokenizácia, respektíve lexikálna analýza). Token je postupnosť znakov, ktoré majú nejaký význam. Najčastejšie sú takýmito lexikálnymi jednotkami samotné slová. Tieto lexikálne jednotky sa musia previesť na výsledné termy, ktoré budú použité pri reprezentácii dokumentu. Tento prevod môže byť veľmi jednoduchý, napríklad lexikálne jednotky budú prevedené na termy konvertovaním všetkých znakov na malé písmená, ale môže takisto pozostávať z viacerých zložitejších krokov. Tokeny sú postupne spracovávané a podľa ich obsahu sú vykonávané príslušné akcie.

3.3.3 Syntaktická analýza

Súbory dokumentov Reuters obsahujú formátovacie tagy, ktoré musia byť spracované. Pokiaľ sa na riadku vyskytuje sekvencia znakov <TEXT TYPE="BRIEF" je to znakom toho, že tento dokument obsahuje síce tagy <TITLE> a </TITLE> avšak neobsahuje tagy <BODY> a </BODY> čo by mohlo viesť k nekonzistencii medzi dvoma výslednými súbormi obsahujúcimi obsah a popis (viď. vyššie). Preto je do reťazca, ktorý uchováva obsah dokumentov medzi tagmi <BODY> A </BODY> vložené označenie transakcie (napr. T542) nasledované znakom konca riadku.

Keď token obsahuje značku <TITLE> do reťazca, ktorý uchováva obsah dokumentu medzi značkami <TITLE> a </TITLE> je vložené označenie transakcie. Následne je značka <TITLE> z tokenu odstránená a na zvyšok z tokenu, ktorý vnikol po odstránení ako aj na všetky nasledujúce tokeny až po výskyt tagu </TITLE> je volaná samotná funkcia predspracovania. Po jej vykonaní sú predspracované tokeny ukladané do reťazca na to určeného (viď. vyššie).

Podobný proces je vykonávaný s tokenmi medzi značkami <BODY> a </BODY>, na ktoré je takisto aplikovaná funkcia predspracovania výsledné tokeny sú uchovávané v reťazci určenom na uchovanie textového obsahu dokumentov. Oba reťazce sú na konci predspracovanie uložené do súborov na to určených (viď. 3.3)

3.3.4 Funkcia na predspracovanie

Funkcia predspracovania, ktorá je volaná na každý token dokumentu vyskytujúci sa medzi tagmi <BODY> a <BODY>, <TITLE> a </TITLE> pozostáva z niekoľkých častí. Najskôr dôjde k nahradeniu HTML značiek „<“ za „<“ a „&“ za „&“. V druhej fáze dôjde k prevodu všetkých písmen na jednotnú veľkosť, a to na malé písmená (okrem anglických názvov mesiacov).

V ďalšej časti sú odstránené interpunkčné znamienka (bodky, čiarky, otázniky) a celkovo znaky, ktoré nie sú alfabetické („úvodzovka“, „(“, „<“, „)“, „>“, „apostrof“, „–“). Najzložitejšou časťou je aplikovanie Porterovho stemmovacieho algoritmu na každý token. Takto upravené tokeny sú v poslednej fáze porovnané so slovníkom nepotrebných slov a v prípade výskytu daného tokenu vo slovníku je toto slovo takzvane zahodené.

3.3.5 Porterov stemmovací algoritmus

Porterov stemmovací algoritmus (alebo „Porterov stemmer“) je proces odstraňovania prostých morfologických a skloňovaných koncoviek z anglických slov. Jeho hlavné využitie spočíva v normalizácii termov ako súčasti procesu automatického vyhľadávania informácií[15].

Algoritmu je založený na idey, že všetky prípony angličtiny (približne 1200) sú väčšinou zložené z kombinácie menších a jednoduchších prípon. Stemmer pracuje lineárnym spôsobom, pozostávajúcim zo šiestich krokov na ktoré sú aplikované pravidlá. V každom kroku sa zisťuje, či suffixové pravidlo je aplikovateľné na dané slovo. Pokiaľ tomu tak je, tak sú testované podmienky vzťahujúce sa k tomuto pravidlu a zisťuje sa, aký by bol výsledný stem, pokiaľ by bol suffix odstránený. Príkladom takého pravidla môže byť počet samohlások, ktoré sú nasledované spoluhláskou v danom slove (= Measure = Veľkosť), ktorá musí byť väčšia než jedna, aby mohlo byť pravidlo aplikovateľné.

Pokiaľ sú splnené podmienky stanovené pravidlom, akcia naviazaná na toto pravidlo je vykonaná, teda daná prípona je odstránená a riadenie je predané do ďalšieho kroku. Pokiaľ pravidlo nie je akceptované, nasledujúce pravidlo v danom kroku je otestované, až pokiaľ nie je splnené nejaké pravidlo v danom kroku, a riadenie je predané do ďalšieho kroku, alebo už v danom kroku neexistujú ďalšie pravidlá a riadenie je automaticky predané do ďalšieho kroku. Tento proces pokračuje pre všetkých šesť krokov, výsledný stem je vrátený zo Stemmera po tom, čo riadenie bolo predané z kroku šesť[16].

Originálny Porterov stemmovací algoritmus je značne agresívny k niektorým typom slov. Zatiaľ čo napríklad slová „connect“, „connected“, „connecting“, „connection“, connestions“ upraví správne na základ „connect“, tak napríklad slovo „adding“ zmení na „ad“, „since“ na „sinc“, „security“ na „secur“, „certificates“ na „certif“ atď. Z týchto dôvodov bolo nutné originálny algoritmus pozmeniť, a to väčšinou ad-hoc spôsobom pre jednotlivé prípady. Kroky a pravidlá algoritmu sú uvedené v prílohe 1.

3.3.6 Slovník nepotrebných slov

Veľa slov v dokumente vôbec nepopisuje jeho obsah (častice, spojky, príslovky). Jedným z krokov v transformácii lexikálnych jednotiek na termy je odstránenie týchto nevýznamných slov, čím sa zníži počet termov. Odoberajú sa aj tie termy, ktoré sa vyskytujú príliš často (v celej množine

dokumentov), môžu byť označené ako nevýznamné respektíve málo významné. Taktiež sa používa takzvaný stoplist - zoznam slov, ktorý obsahuje bežné nevýznamové slová (stop words, stopslová). Vo všeobecnosti pomocou stoplistu sa dá odstrániť 40 - 50% z celkového počtu slov.

Slovník nepotrebných slov obsahuje 130 slov, ktoré sa vyskytovali v dokumentoch často a nesú žiaden alebo príliš malý význam. Jedná sa najmä o vetné členy, spojky, predložky, príslovky, zámená, atď. Takéto slová sa vyskytovali v asociačných pravidlách často, avšak tieto asociačné pravidlá boli pre získavanie znalostí nepoužiteľné. Slovník je vlastne textový súbor, kde je každé slovo na samostatnom riadku. Slová sú usporiadané abecedne čo umožňuje rýchlejšie vyhľadávanie v slovníku použitím funkcie využívajúcej vlastnosti binárneho stromu. Použitím slovníka bolo do výstupného súboru zaradených o 44% menej slov ako pred použitím slovníka. Ukážka slovníka sa nachádza v prílohe 2.

4 Asociačné pravidlá

4.1 Definícia asociačných pravidiel

Dolovanie asociačných pravidiel v transakčných databázach bolo dokázané ako prospešné a technicky uskutočniteľné v niekoľkých aplikačných oblastiach, najmä v obchodnej sfére.

Nech $I = \{i_1, i_2, i_3,..., i_m\}$ je množina položiek. Nech D je množina transakcií, kde každá transakcia T je množina položiek taká, že $T \subseteq I$. Asociačné pravidlo je implikácia vo forme $X \Rightarrow Y$, kde $X \subset I$, $Y \subset I$ a $X \cap Y = \emptyset$. Asociačné pravidlo $X \Rightarrow Y$ má podporu s, pokiaľ s% transkakcií v D obsahuje $X \cup Y$ a spoľahlivosť c, pokiaľ c% transkakcií v D, ktoré obsahujú X obsahujú zároveň aj Y. Dolovanie asociačných pravidiel znamená hľadanie všetkých asociačných pravidiel, ktoré majú podporu a spoľahlivosť väčšiu alebo rovnú zvolenej minimálnej podpore, nazvanej minsup, a zároveň minimálnej spoľahlivosti, nazvanej minconf (prevzaté z [5]).

Príkladom z praxe môžu byť položky nákupného košíka. Vezmime si napríklad tovary ako pivo a plienky. Pivo $\Rightarrow$ plienky je asociačné pravidlo vydolované z databáze, pokiaľ spoločný výskyt piva a plienok (v rovnakej transakcii) nie je menší než minsup a výskyt piva v transakciách obsahujúcich plienky nie je menší než minconf.

Prvým krokom v získavaní asociačných pravidiel je získanie všetkých množín položiek takých, ktoré majú mieru spoločného výskytu vyššiu ako je minimálna podpora. Takéto množiny sa nazývajú frekventované množiny. Veľkosť množiny je daná počtom položiek množiny, a teda množina obsahujúca k položiek sa bude nazývať k-množina. Napríklad: množina {pivo, plienky} môže byť frekventovaná 2-množina. Nájdenie všetkých frekventovaných množín je veľmi náročná úloha na zdroje, ktorá v poslednom období zaznamenala značné množstvo bádania.

Druhý krok pri vytváraní asociačných pravidiel z frekventovaných množín prebieha podľa nasledujúceho postupu: pre každú množinu f nájdi všetky neprázdne podmnožiny f. Pre každú takúto podmnožinu a vytvor pravidlo v tvare $a \Rightarrow (f - a)$ pokiaľ pomer podpory $(f - a)$ k podpore (a) je prinajmenšom rovné minconf.

4.2 Využitie asociačných pravidiel

Asociačné pravidlá vydolované z obchodného hľadiska transakčných databáz môžu byť použité na predpovedanie nakupovania zákazníkov. V textových databázach existuje niekoľko použití vydolovaných asociačných pravidiel.

Môžu byť použité napríklad na vytvorenie štatistického slovníka. Uvažujme prípad, že máme asociačné pravidlo $B \Rightarrow C$, kde B a C sú slová. Hľadanie dokumentu obsahujúceho C môže byť

rozšírené o pridanie B. Toto rozšírenie nám umožní hľadanie takých dokumentov pomocou C, ktoré neobsahujú C ako pojem.

Úzko súvisiace použitie je indexácia latentnej sémantiky (Latent Semantic Indexing), kde dokumenty sú považované za navzájom blízke, pokiaľ zdieľajú dostatočný počet asociácií. Indexácia latentnej sémantiky môže byť použitá na získanie dokumentov, ktoré nemajú žiadne spoločné pojmy s originálnym hľadaným výrazom, a to pridaním dokumentov do množiny výsledkov dotazu, ktoré sú blízko originálnej množiny výsledkov dotazu.

Rozloženie početnosti textovej databáze môže byť veľmi odlišné od rozloženia početnosti položiek v transakčnej databáze. Navyše počet jednoznačných slov v textovej databáze je značne vyšší, než počet položiek v transakčnej databáze. A nakoniec počet jedinečných slov v typickom dokumente je omnoho vyšší, než počet unikátnych položiek v transakciách. Tieto skutočnosti sú dôvodom, prečo sú klasické algoritmy ako Apriori a Direct Hashing and Pruning (DHP) neefektívne v dolovaní asociačných pravidiel v textových databázach.

4.3 Získavanie asociačných pravidiel z textov

Tradičným oborom hľadania frekventovaných množín a následne asociačných pravidiel sú obchodné transakčné databázy a databázy katalógu objednávok. Zvyčajnými položkami vyskytujúcimi sa v transakciách sú transakčné položky, avšak môžu byť prítomné aj iné typy, napríklad individuálna história zákazníka. Taká položka môže mať detailnú identitu, ako napríklad určitý druh chleba, veľkosť, alebo naopak môže byť namapovaná na všeobecnú identitu ako napríklad „chlieb“. Počet položiek v typickej obchodnej transakcii je značne pod sto. Stredná hodnota položiek a rozloženia je značne závislá na obchodnej operácii. Experimentálnym zisťovaním sa prišlo na počet položiek v transakciách v rozmedzí od 5 do 20.

Charakteristika rozloženia slov v textových dátach znamená niektoré veľké výzvy pre algoritmy, ktoré sú typicky používané na obchodné transakčné databáze. Uveďme si ako príklad databázu textových dokumentov Wall Street Journal z roku 1990, ktoré boli uvedené v súbore dát TReC. Súbor obsahoval 3568 článkov a 47189 unikátnych slov. Väčšina slov sa vyskytovala iba v niekoľkých dokumentoch. Stredná hodnota unikátnych slov v dokumente bola 207 so štandardnou odchýlkou 174,2 slov. V skúmanej vzorke dát iba 6,8% slov sa vyskytlo vo viac ako v 1% dokumentov.

Charakteristika tohto rozloženia slov má nesmierny dopad na efektivitu algoritmu získavania asociačných pravidiel. Najdôležitejšie dopady sú: 1) veľké množstvo položiek a kombinácií položiek, ktoré musia byť spočítané a 2) veľký počet položiek v každom dokumente databázy.

Je známe, že slová, ktoré sa vyskytujú rovnomerne v textových databázach majú malú hodnotu v rozlišovaní dokumentov a navyše tieto slová sa vyskytujú v značnom počte dokumentov. Je rozumné predpokladať, že frekventované množiny zložené z vysoko frekventovaných slov (typicky

s mierou výskytu nad 20%) budú mať takisto malú hodnotu. Preto pre dolovanie dát sú najvhodnejšie množiny zložené zo slov, ktoré nie sú príliš frekventované, ale na druhú stranu sú frekventované dostatočne. Presný rozsah minimálnej a maximálnej podpory vhodnej pre získavanie asociačných pravidiel nie je známy, avšak je jasné, že minimálna podpora bude omnoho menšia, než typicky používaná pri obchodných transakčných databázach.

Relatívne nízka minimálna podpora vyžadovaná pre dolovanie dát z textu ešte zhoršuje problém daný frekvenčným rozložením slov. V textových databázach sa prevažná časť slov vyskytuje so strednou alebo nízkou mierou výskytu a tieto slová sú presne tie, ktoré sú pre hľadanie frekventovaných množín zaujímavé.

Nie je jasné, aká by mala byť minimálna podpora pre hľadanie efektívnych asociácií. Experimentálne výsledky ukázali, že podpora by mala byť nižšia ako 0,5% pre kolekciu dát Wall Street Journal z apríla 1990 a ako nakoniec bola vybraná minimálna podpora 0,1%.

4.3.1 Asociačné pravidlá získané z kolekcie dát Reuters-21578

V tejto časti budú ukázané asociačné pravidlá, ktoré vznikli z dokumentov nachádzajúcich sa v súbore reut2-002.sgm. Tento súbor, tak ako ostatné, obsahuje 1000 dokumentov nad ktorými bolo najskôr vykonané predspracovanie, potom získavanie frekventovaných množín a z nich nakoniec získavané asociačné pravidlá.

Z dôvodu veľkého počtu dokumentov bola zvolená hranica minimálnej podpory na 0,5%. Aj pri tejto o trochu vyššej hodnote, ako je odporúčaná hodnota v literatúre bolo získaných 2609 frekventovaných 1-množín, z ktorých bolo vygenerovaných 3,4 milióna kandidátnych 2-množín. Podmienku minimálnej podpory splnilo 13381 množín a tie tvoria frekventované 2-množiny. Frekventovaných 3-množín bolo 28328, frekventovaných 4-množín 22434, frekventovaných 5-množín 13582, frekventovaných 6-množín 8066, frekventovaných 7-množín 4788, frekventovaných 8-množín 2515, frekventovaných 9-množín 1030, frekventovaných 10-množín 294 a konečne frekventovaných 11-množín 51. Ako už z jednotlivých počtov frekventovaných k-množín vyplýva, nie je možné vypísať všetky asociačné pravidlá, preto sa zameriam iba na niektoré zaujímavé a uvediem aj ich mieru spoľahlivosti (viac asociačných pravidiel sa nachádza v prílohe 3):

asociačné pravidlo	**spoľahlivosť (confidence)**
kuwaiti, response => attack, iran, iranian, u.s.	100%
attack, iran, iranian, ship => monday, u.s.	80%
oil, quota, report, stock => production	100%
stop, u.s., world => market, policy, subsidy, support	100%
market, reform, subsidy, world => policy, support, trade	100%
market, rate, trade, unit => many, surplus	100%
need, support, trade => u.s., world	66%

5 Algoritmy získavania asociačných pravidiel v textových dátach

V tejto kapitole predostriem nový algoritmus na dolovanie asociačných pravidiel medzi slovami v textových databázach.

Charakteristika textových databáz je značne odlišná od obchodných transakčných databáz pre ktoré sa dolovanie dát väčšinou robí. Existujúce dolovacie algoritmy teda nemusia zvládnuť textové databáze efektívne z dôvodu priveľkého množstva položiek (t.j. slov) nad ktorými sa dolovanie robí. Dobre známy dolovací algoritmus Apriori bude konfrontovaný s novým algoritmom nazvaným Multipass-Apriori (M-Apriori) v kontexte dolovania textových databáz.

5.1 Existujúce algoritmy pre dolovanie asociačných pravidiel

Algoritmy na hľadanie frekventovaných množín vo väčšine prípadov využívajú viacnásobného priechodu databázou. V prvom priechode je spočítaná podpora individuálnych množín a sú nájdené 1-množiny. Potom 1-množny sú použité na vygenerovanie potenciálnych 2-množín, ktoré sa nazývajú kandidátne 2-množiny. V druhom priechode spočítame podporu kandidátnych 2-množín a určíme frekventované 2-množiny. Frekventované 2-množiny sú použité na vygenerovanie kandidátnych 3-množín a tak ďalej. Tento proces sa opakuje, až nie je možné vytvoriť žiadne ďalšie frekventované množiny. Existujú sekvenčné ako aj paralelné algoritmy, ktoré sa používajú na získanie frekventovaných množín v transakčných databázach. Algoritmy hľadania frekventovaných množín majú sekvenčnú ako aj paralelnú verziu, ktorá vznikne zmenou sekvenčnej implementácie.

Každý z algoritmov je implementovaný ako proces, ktorý vzájomne pôsobí s lokálnym súborom dát. V paralelných verziách je tento súbor dát rozdelený na podsúbory, pričom každý z nich sa stáva lokálnym pre daný procesor. Existujú aj alternatívne prístupy, ktoré vyžadujú presun všetkých alebo podstatnej časti dát z lokálneho prostredia do ostatných prostredí na paralelné spracovanie. Tieto metódy neboli uskutočnené, pretože vyžadujú značné množstvo presunov a replikácií dát. Kolekcie textov lokálnych pre daný procesor môžu mať aj niekoľko gigabytov, a teda nie sú vhodné na prenos a replikáciu medzi desiatky alebo stovky procesorov.

5.1.1 Algoritmus Apriori

Algoritmus Apriori navrhnutý Agrawalom a Srikantom na hľadanie frekventovaných množín, kde vstupné dáta pozostávajú z transakcií je nasledovný[5]:

1) Database = množina transakcií;
2) Items = množina položiek;
3) transaction = <TID, {x | x ∈ Items}>;
4) *Poznámka:* F_1 je množina frekventovaných 1-množín
5) $F_1 = \emptyset$;
6) *Poznámka:* Prejdi transakcie a spočítaj výskyty všetkých položiek;
7) **foreach** transaction t ∈ Database **do begin**
8) **foreach** item x **in** t **do**
9) x.count++;
10) **end**
11) *Poznámka:* Vytvor množinu frekventovaných 1-monožín
12) **foreach** item i ∈ Items **do**
13) **if** i.count / |Database| ≥ minsup **then**
14) $F_1 = F_1 \cup i$;
15) *Poznámka:* Nájdi F_k, množinu frekventovaných k-množín, kde $k \geq 2$
16) **for** (k := 2; $F_{k-1} \neq \emptyset$; k++) **do begin**
17) *Poznámka:* C_k je množina kandidátnych k-množín
18) $C_k = \emptyset$;
19) *Poznámka:* $F_{k-1} * F_{k-1}$ je prirodzené spojenie F_{k-1} a F_{k-1} prvých k – 2 množín
20) **foreach** $x \in \{ F_{k-1} * F_{k-1}\}$ **do**
21) **if** $\neg\exists y \mid y$ = (k - 1)-podmnožina $x \wedge y \notin F_{k-1}$ **then**
22) $C_k = C_k \cup x$;
23) *Poznámka:* Prehliadni transakcie a spočítaj kandidátne k-množiny
24) **foreach** transaction t ∈ Database **do begin**
25) **foreach** k-množinu x **in** t **do**
26) **if** $x \in C_k$ **then**
27) x.count++;
28) **end**
29) *Poznámka:* F_k je množina frekventovaných k-množín
30) $F_k = \emptyset$;
31) **foreach** $x \in C_k$ **do**
32) **if** x.count / |Database| ≥ minsup **then**

33) $F_k = F_k \cup x$;

34) **end**

35) výsledok = $\cup_k F_k$;

Vytváranie kandidátnych množín môže byť efektívnejšie, keď sú položky v každej množine uložené v lexikálnom poradí a samotné množiny sú takisto lexikálne usporiadané. Ako je uvedené na riadku 20 kandidátne k-množiny pre k ≥ 2 sa získajú pomocou operácie prirodzeného spojenia F_{k-1} * F_{k-1} na prvých k – 2 množinách F_{k-1} za predpokladu, že položky sú vo všetkých množinách lexikálne usporiadané.

Na príklade si uveďme ako dôjde k vytváraniu kandidátnych množín. Pokiaľ napríklad F_2 = ({A, B}, {A, C}), potom {A, B, C} je potenciálna kandidátna 3-množina. Na riadku 21 dôjde k obmedzeniu kandidátnych k-množín s využitím vlastnosti, že všetky (k-1)-množiny frekventovaných k-možín musia byť frekventované (k-1)-množiny. Táto vlastnosť sa nazýva množinová uzáverová vlastnosť frekventovaných množín. A teda pokiaľ má byť množina {A, B, C} kandidátnou 3-množinou, množina {B, C} by takisto mala byť frekventovanou 2-množinou.

Tento obmedzujúci krok zamedzí zaradeniu mnohých potenciálnych množín do kandidátnych množín, čo má za následok veľké zníženie nárokov na pamäť. Na efektívne počítanie výskytov kandidátnych množín v transakciách sa vo veľkej miere využíva štruktúra hashovacieho stromu, kde hash hodnota každej položky sa nachádza na jednej úrovni v strome.

Algoritmus Apriori môže byť paralelizovaný jednoduchým spôsobom, a to distribúciou transakcií procesorom a nazdieľaním množín na konci každého priechodu. Nanešťastie množstvo textu, ktoré je vyhradené pre procesor môže stále byť príliš veľké a môže generovať veľmi veľké počty kandidátnych množín pre algoritmus Apriori. Pripomeňme si charakter rozloženia kolekcie dát z kapitoly 4.3. Bude sa v nej nachádzať značné množstvo kandidátnych množín, ktoré nemajú minimálnu podporu. V danom súbore dát bolo približne 15000 slov, ktoré sa vyskytujú vo viac než 0,1% dokumentov. Pomocou algoritmu Apriori bude vygenerovaných 112 miliónov kandidátnych 2-množín.

5.1.2 Algoritmus Multipass-Apriori

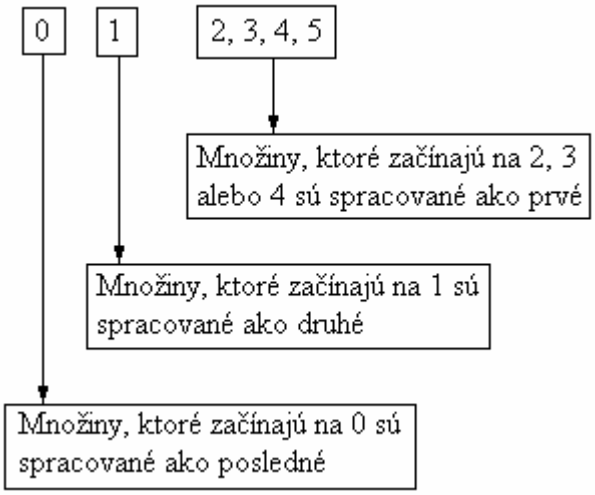
Algoritmus Multipass-Apriori (M-Apriori) pre dolovanie asociačných pravidiel je priamym nasledovníkom algoritmu Apriori. Ako bolo poukázané v predchádzajúcej stati, algoritmus Apriori, nie je vhodný pre získavanie asociačných pravidiel z textových databáz, z dôvodu veľkých pamäťových nárokov na spočítanie výskytov veľkého počtu potenciálnych frekventovaných množín. Prístup viacnásobného priechodu (Multipass) priamo redukuje potreby na pamäť, a to rozdelením frekventovaných 1-množín a ich osobitným spracovaním. Každá partícia obsahuje časť položiek z celej množiny databáze, takže veľkosť pamäte potrebnej na spočítanie výskytov množín v rámci danej partície bude oveľa menšia, než v prípade spočítania výskytov všetkých položiek v databázy.

Algoritmus M-Apriori môže byť popísaný nasledujúcim spôsobom[5]:

1. Je spočítaný počet výskytov každej položky databáze a sú nájdené frekventované 1-množiny.
2. Frekventované 1-množiny sú rozdelené na p častí: P_1, P_2, ..., P_p
3. Použijeme algoritmus Apriori na nájdenie všetkých frekventovaných množín v každej partícii v poradí P_p, P_{p-1},...,P_1. Keď je prehľadávaná partícia P_p, môžeme nájsť všetky frekventované množiny, ktorých členské položky sú v P_p. Keď je spracovávaná ďalšia partícia P_{p-1}, môžeme nájsť všetky frekventované množiny, ktorých členské položky sú v P_{p-1} a v P_p. Je to z dôvodu, že keď je spracovávané P_{p-1}, frekventované množiny, ktoré sme našli v P_p sú rozšírené položkami z P_{p-1} a potom spočítané. Takto to pokračuje, až nie je spracovaná partícia P_1.

Bez straty všeobecnosti uvažujme, že položky sú usporiadané lexikálne. Položky sú rozdelené na p partícií, P_1, P_2, ..., P_p, takže pre každé i < j, každá položka a $\in$ P_i je menšia než každá položka b $\in$ P_j. Všimnime si, že pokiaľ by partície mali rovnaký počet prvkov, potenciálny počet položiek, ktoré by boli sformované rozšírením lexikálne nižšie zoradených partícií by bol väčší, než potenciálny počet položiek z lexikálne vyššie zoradených partícií.

Vzhľadom k tomu, že položky sú usporiadané lexikálne, je dôležité spracovávať partície v poradí od najvyššie zoradených po najnižšie zoradené. Toto poradie spracovania je požadované na podporenie prerezania kandidátnych množín založenom na uzáverovej vlastnosti frekventovaných množín. Obrázok 5.1 ukazuje príklad partícií, kde položky 0, 1, 2, 3, 4 a 5 sú frekventované 1-množiny a sú rozdelené na P_1, P_2 a P_3.



Obr. 5.1: Rozdelenie množiny šiestich frekventovaných položiek pre Multipass-Apriori

Napríklad, keď je zistené, že množina {2, 3} je frekventovaná ako výsledok spracovania položiek v partícii P3, potom môžeme spočítať položky {1, 2, 3} keď je partícia P_2 spracovávaná.

V praxi, pokiaľ je odhadovaný počet kandidátnych množín, ktoré sú generované po spracovaní určitého počtu partícií malý, potom môžeme spojiť zvyšné partície do jednej, a tým zredukovať počet priechodov databázy.

6 Porovnanie efektivity algoritmov Apriori a Multipass-Apriori

Pred tým, než budú porovnané obidva algoritmy a ich vhodnosť pre získavanie asociačných pravidiel z dát, uvedieme si na konkrétnom prípade princíp fungovania algoritmu Multipass-Apriori.

6.1 Príklad fungovania Multipass-Apriori

Na nasledujúcom konkrétnom príklade si ukážeme, ako funguje algoritmus Multipass-Apriori.
Majme nasledujúcu databázu a minimálnu podporu minsup = 3.

T1 cokolada ryba konzerva syr
T2 ryba syr olivy
T3 syr sol
T4 ryba konzerva
T5 syr cokolada ryba konzerva olivy
T6 sol
T7 keksy ryba med
T8 keksy salama jablko
T9 jablko konzerva
T10 maslo chlieb rozok maso
T11 syr kremes maslo maso
T12 syr maslo rozok kremes maso
T13 maslo konzerva dzem
T14 rozok maso maslo chlieb olivy
T15 dzem maslo rozok
T16 maso rozok maslo
T17 keksy rozok jablko
T18 maso konzerva

1. krok

Získame kandidátnu 1-množinu spolu s podporou. C_1 = { chlieb:2, cokolada:2, dzem:2, jablko:3, keksy:3, konzerva:6, kremes:2, maslo:7, maso:6, med:1, olivy:3, rozok:6, ryba:5, salama:1, sol:2, syr:6 }

Podľa podpory, ktorá je uvedená za každým prvkom množiny C_1 určíme tie, ktoré majú podporu vyššiu alebo rovnú 3. Tieto prvky presunieme do množiny frekventovaných jednoprvkových množín. F_1 = { jablko:3, keksy:3, konzerva:6, maslo:7, maso:6, olivy:3, rozok:6, ryba:5, syr:6 }

Tento krok je úplne rovnaký ako pri Apriori algoritme.

2. krok

Vypočítame počet prvkov P_p časti a tieto prvky spracujeme Apriori algoritmom. Do partície P_p dáme teda z F_1 položky P_p = {maso:6, olivy:3, rozok:6, ryba:5, syr:6}.

Najskôr sú nájdené kadidátne 2-množiny C_2 = { {maso, olivy}:1, {maso, rozok}:4, {maso, ryba}:0, {maso, syr}:2, {olivy, rozok}:1, {olivy, ryba}:2, {olivy, syr}:2, {rozok, ryba}:0, {rozok, syr}:1, {ryba, syr}:3 }. Z nich podmienku minimálnej podpory spĺňajú iba množiny { {maso, rozok}:4, {ryba, syr}:3 } ktoré tvoria frekventované 2-množiny. Z nich nie je možné vytvoriť kandidátne 3-množiny.

3. krok

Vypočítame počet prvkov P_{p-1} časti a vložíme ich do partície P_{p-1} = {konzerva, maslo}.

Kandidátne 2-množiny (C_2) získame kombináciou prvkov z množiny P_{p-1} a k nim pridáme množiny získané kartézskym súčinom prvkov z P_{p-1} a prvkov z P_p. C_2 = { {konzerva, maslo}:1, {konzerva, maso}:1, {konzerva, olivy}:1, {konzerva, rozok}:0, {konzerva, ryba}:3, {konzerva, syr}:2, {maslo, maso}:5, {maslo, olivy}:1, {maslo, rozok}:5, {maslo, ryba}:0, {maslo, syr}:2 }.

Z množiny C_2 spĺňajú podmienku minimálnej podpory iba množiny F_2 = { {konzerva, ryba}:3, {maslo, maso}:5, {maslo, rozok}:5 }. Z nich sa dá vytvoriť kandidátna 3-množina, ktorá je zároveň frekventovanou 3-množinou, pretože spĺňa podmienku minimálnej podpory $C_3 = F_3$ = { {maslo, maso, rozok}:4 }.

4. krok

Vypočítame počet prvkov P_{p-2} časti a vložíme ich do partície P_{p-2} = {jablko, keksy}.

Opäť získame kandidátnu 2-množinu (C_2) kombináciou prvkov z množiny P_{p-2} a kartézskym súčtom prvkov z množiny P_{p-2} a zjednotením množín P_{p-1} a P_p. C_2 = { {jablko, keksy}:2, {jablko, konzerva}:1, {jablko, maslo}:0, {jablko, maso}:0, {jablko, olivy}:0, {jablko, rozok}:1, {jablko, ryba}:0, {jablko, syr}:0 , {keksy, konzerva}:0, {keksy, maslo}:0, {keksy, maso}:0, {keksy, olivy}:0, {keksy, rozok}:1, {keksy, ryba}:1, {keksy, syr}:0 }. Podmienku minimálnej podpory nespĺňa žiadna z množín, a v tomto kroku teda zostáva množina F_2 prázdna.

Algoritmus končí a výsledkom sú všetky k-prvkové F_k množiny zo všetkých krokov.

2-prvkové: { {konzerva, ryba}, {maslo, maso}, {maslo, rozok}, {maso, rozok}, {ryba, syr} }

3-prvková: { {maslo, maso, rozok} }

6.2 Voľba veľkosti partícií

Rozhodujúcim faktorom ovplyvňujúcim efektivitu Multipass-Apriori algoritmu je voľba veľkosti partícií. Veličiny, ktoré budú skúmané sú časová a pamäťová náročnosť. Špecifikom Multipass prístupu je hľadanie kandidátnych množín z frekventovaných množín po častiach (partíciách), je teda predpoklad, tento prístup bude oproti klasickému Apriori z pamäťového hľadiska efektívnejší.

Z časového aj pamäťového hľadiska je najnáročnejším krokom algoritmu hľadanie 2-prvkových kandidátnych množín z 1-prvkových frekventovaných množín. Napríklad pre 155 prvkov frekventovaných 1-množín vznikne 11935 kandidátnych 2-množín, z ktorých podmienku minimálnej podpory splní iba 10 a z nich vznikne už len 1 frekventovaná 3-množina. Tento príklad sa dá zovšeobecniť, a usúdiť, že o pamäťovej náročnosti v hlavnej miere rozhoduje počet vygenerovaných kandidátnych 2-množín.

Na výpočet počtu kandidátnych 2-množín z N frekventovaných položiek (1-množín) použijeme vzorec[11]:

$$\frac{N^2 + N}{2}$$

Vzorec 6.1

Existuje niekoľko prístupov, ako voliť veľkosť partícií. Sú to:

- Lineárne rozdelenie
- Inkrementálne rozdelenie
- Exponenciálne rozdelenie
- Rozdelenie zamerané na rovnaké (veľmi podobné) počty kandidátnych 2-množín ktoré vznikli pridaním novej partície so zameraním na stanovený počet partícií (striktný a benevolentný variant)
- Rozdelenie zamerané na rovnaké (veľmi podobné) počty kandidátnych 2-množín ktoré vznikli pridaním novej partície so zameraním na pomerové vyjadrenie počtu kandidátnych 2-množín v jednotlivých partíciách k celkovému počtu kandidátnych 2-množín

V nasledujúcej časti budú popísane jednotlivé prístupy a na konkrétnych testoch bude ukázané, ktoré metódy majú aké vlastnosti a ktorý z prístupov je najefektívnejší. Kritéria, podľa ktorých boli jednotlivé prístupy hodnotené boli: čas výpočtu, najväčší kus pamäte alokovaný počas výpočtu, rozptyl počtu kandidátnych množín vzniknutých pri spracovaní jednotlivých partícií.

Počítač na ktorom boli testy vykonané mal takúto konfiguráciu: AMD Athlon XP 1800+ (1,53GHz), 768MB DDR RAM 333Mhz.

6.2.1 Lineárne rozdelenie partícií

Prvotná myšlienka na rozdelenie frekventovaných 1-množín na partície je rozdeliť ich lineárne. To značí snažiť sa rozdeliť množinu 1-množín na približne rovnaký dopredu stanovený počet množín a z nich vytvárať pomocou metódy Multipass-Apriori kandidátne množiny a z nich následne frekventované množiny.

Pokiaľ necháme počet partícií nastavený na hodnotu 1, dostávame vlastne klasickú metódu Apriori. Po jej vykonaní bolo nájdených 11935 kandidátnych 2-množín, veľkosť pamäte zabranej pri výpočte bola 4362,800KB a doba výpočtu bola 48,201s. Ako bude neskôr ukázané, množstvo

využitej pamäte niekoľkonásobne prevyšuje množstvo pamäte potrebnej pre ktorýkoľvek z prístupov delenia partícií za použitia Multipass algoritmu.

Celý proces rozdeľovania bude zachytený a vysvetlený na nasledujúcom príklade. Praktické testy pre túto aj ostatné metódy boli vykonané na vzorke dát, ktorá obsahovala 155 frekventovaných 1-množín, z ktorých je možné vytvoriť 11935 kandidátnych 2-množín.

Rozdelenie frekventovanej 1-množiny (ktorá má spomínaných 155 prvkov) na 2 časti: 79 a 76. Z prvej časti bolo vytvorených 3081 kandidátnych množín a z druhej 8854 (z toho 6004 kartézskym súčinom prvkov prvej a druhej partície a 2850 vytvorených podľa vzorca 6.1 v rámci danej partície). Čas výpočtu bol 49,422s a najväčší kus alokovanej pamäte bol 1735,616KB. Smerodajná odchýlka určená pre počty kandidátnych 2-množín týchto partícií bola 2886,5. Pri delení na 3 časti: 54, 51 a 50 vzniklo 1431, 4029 respektíve 6475 kandidátnych 2-množín. Čas výpočtu bol 49,235s, najväčší kus alokovanej pamäte bol 1325,156KB a smerodajná odchýlka pre počty kandidátnych 2-množín v jednotlivých partíciách bola 2059,5. Namerané hodnoty pre počet partícií 2 až 9 spolu s uvedenými počtami kandid. 2-množín, ktoré vzniknú spracovaním jednotlivých partícií sú v nasled. tabuľkách:

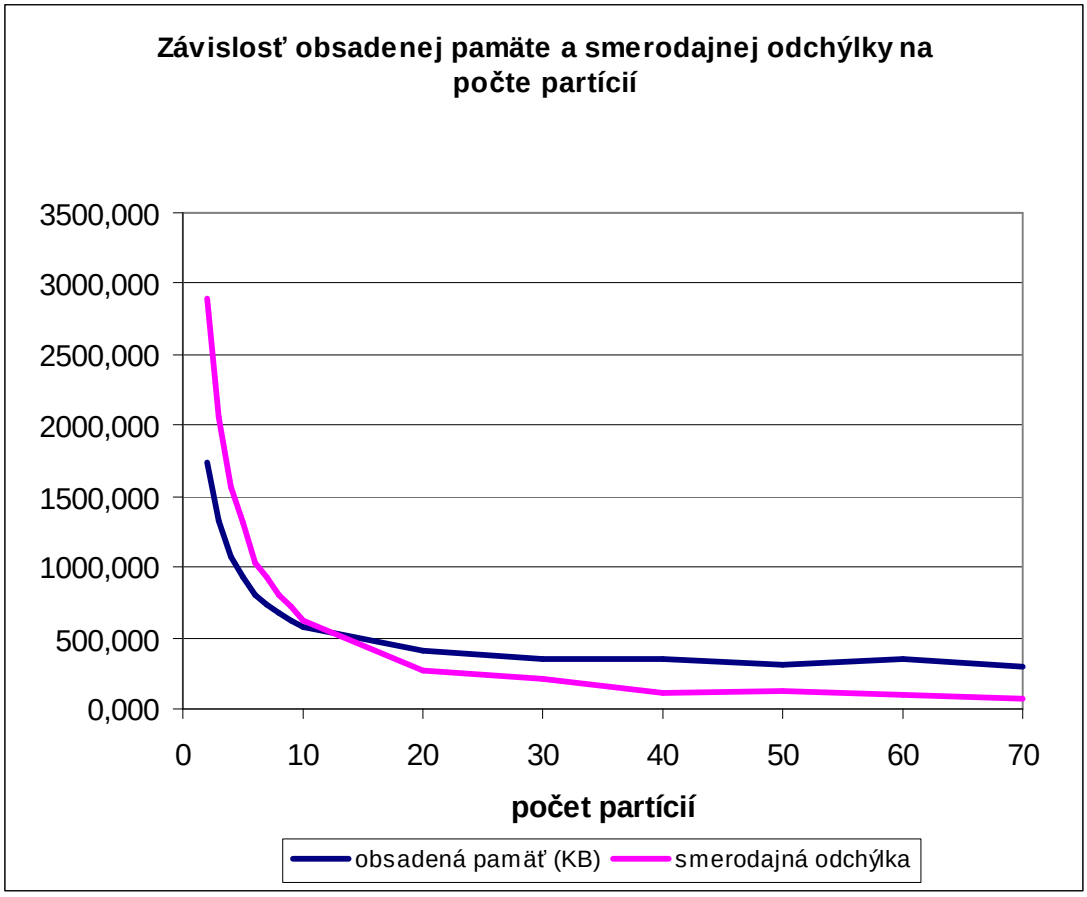
počet partícií		**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
kandidátne 2-množiny v jednotlivých partíciách	Pp	3081	1431	861	496	465	276	253	190
	Pp-1	8854	4029	2299	1457	1075	759	608	476
	Pp-2		6475	3743	2418	1700	1243	969	765
	Pp-3			5032	3379	2325	1727	1330	1054
	Pp-4				4185	2950	2211	1691	1343
	Pp-5					3420	2695	2052	1632
	Pp-6						3024	2413	1921
	Pp-7							2619	2210
	Pp-8								2344
obsadená pamäť (KB)		*1735,616*	*1325,156*	*1076,656*	*937,060*	*804,016*	*739,264*	*672,672*	*627,608*
smerodajná odchýlka		*2886,5*	*2059,5*	*1561,0*	*1315,9*	*1026,8*	*935,6*	*797,5*	*720,1*
čas výpočtu (s)		*49,422*	*49,235*	*49,984*	*50,109*	*49,25*	*49,969*	*50,047*	*48,969*

Tabuľka 6.1: Získané hodnoty metódy lineárneho rozdelenia partícií pre počet partícií od 2 do 9

počet partícií	**10**	**20**	**30**	**40**	**50**	**60**	**70**
obsadená pamäť (KB)	*577,784*	*405,664*	*358,232*	*353,915*	*349,416*	*347,712*	*292,296*
smerodajná odchýlka	*620,9*	*262,6*	*208,6*	*119,5*	*125,9*	*93,1*	*77,4*
čas výpočtu (s)	*50,407*	*49,859*	*51,25*	*52,386*	*54,769*	*55,594*	*57,953*

Tabuľka 6.2: Získané hodnoty metódy lineárneho rozdelenia partícií pre počet partícií od 10 do 70

Z hodnôt v tabuľkách sa dá vyčítať, že pamäť obsadená pri výpočte kandidátnych množín je v úzkom vzťahu so smerodajnou odchýlkou počtu týchto kandidátnych množín. Túto skutočnosť zachycuje aj priebeh grafu 6.1.



Graf 6.1: Graf závislosti obsadenej pamäte a smerodajnej odchýlky na počte partícií

Takisto môžeme zistiť, že množstvo času potrebného na výpočet rastie s pribúdajúcim počtom partícií. Táto zmena nie je až taká výrazná, keďže napríklad pri rozdelení na 3 a na 60 partícií, čo je 20 násobné zvýšenie počtu partícií bola časová zmena iba 6,269 sekúnd.

Veľmi podstatný fakt ktorý vystihuje nedostatok tejto metódy je, že vždy v poslednej partícii je vygenerovaný najväčší počet kandidátnych množín zo všetkých partícií. Pamäťová náročnosť pre akékoľvek rozdelenie partícií je daná práve počtom vygenerovaných hodnôt práve v tejto poslednej partícii. Práve z tohto dôvodu nie je opodstatnené ďalšie zvyšovanie počtu partícií, pri ktorom by síce došlo k malému zníženiu pamäťových nárokov, avšak nie ideálne efektívnym spôsobom, a preto je vhodnejšie hľadanie nových metód rozdelenia partícií.

6.2.2 Inkrementálne rozdelenie partícií

Metóda inkrementálneho rozdelenia partícií je vlastne špecifickým prípadom lineárnej funkcie. Pokiaľ počet partícií zvolíme o 1 menej ako je skutočný počet partícií, každá partícia bude obsahovať

1 prvok (okrem partície Pp, ktorá bude obsahovať 2 prvky). Praktickým testom s touto metódou bolo nájdených 154 partícií, najväčší kus obsadenej pamäte bol 280,992KB a výpočet trval 80,938s. Množstvo obsadenej pamäte bolo síce o 11,304KB menšie než najlepší testovaný prípad rozdelenia na 70 partícií v lineárnej metóde, avšak čas výpočtu bol podstatne väčší, a to konkrétne o 23 sekúnd. Tento fakt nás vedie k záveru, že daný spôsob rozdelenia partícií by bol príliš časovo náročný najmä pre väčšie počty kandidátnych 2-množín.

6.2.3 Exponenciálne rozdelenie partícií

Nepríjemný nedostatok lineárnej metódy vo vygenerovaní najväčšieho počtu kandidátnych 2-množín z poslednej partície (P1) sa snaží odstrániť metóda exponenciálneho rozdelenia partícií. Princípom tejto metódy je rozdelenie frekventovanej 1-množiny podľa prednastavenej exponenciálnej hodnoty. Pokiaľ zvolíme za exponent číslo 3, bude mať partícia P1 3 prvky, P2 9 prvkov, P3 27 prvkov, P4 81 prvkov, P5 243 prvkov atď.

Konkrétny príklad rozdelenia frekventovanej 1-množiny na partície pre exponent 4 je uvedený v nasledujúcej tabuľke.

partícia	rozsah indexov	počet prvkov z frekventovanej 1-množiny	počet kandidátnych 2-množín
P4	64 - 155	91	(91^2+91)/2 = 4186
P3	16 - 63	48	(48^2+48)/2 + 48*91 = 5544
P2	4 - 15	12	(12^2+12)/2 + (48+91)*12 = 1746
P1	0 - 3	3	(3^2+3)/2 + (12+48+91)*3 = 459

Tabuľka 6.3: Rozdelenie frekventovanej 1-množiny na partície pre exponent 4

Princíp delenia pozostáva v prvej fáze zo zistenia indexov z exponenciálnej rady. Pre číslo 4 sú to indexy: 1, 4, 16, 64, 256, atď. Číslo 64 je najväčšie menšie číslo než limit daný počtom frekventovaných 1-množín (155). Do partície P4 budú teda pridané prvky od indexu 64 do 155. Celkovo je to 91 prvkov, z ktorých je možné vytvoriť 4186 kandidátnych 2-prvkových množín. Ďalším indexom je prvé menšie číslo než 64 z exponenciálnej rady, a to 16. V partícii P3 bude teda 48 prvkov, z ktorých je možné vytvoriť kombináciou s partíciou P4 4368 (=48*91) plus 1176 (=(48^2+48)/2) v rámci danej partície, spolu teda 5544 kandidátnych 2-množín. Pridaním partície P2 bude vytvorených 1746 nových kandidátnych 2-množín a konečne pridaním partície P1 459 kandidátnych 2-množín. Už aj na tomto konkrétnom prípade je vidieť trend, v akom počte sú kandidátne 2-množiny v jednotlivých partíciách vytvárané, a teda, že oproti lineárnemu rozdeleniu partícií je v partícii P1 najmenej prvkov.

Rozdelenia partícií pre iné exponenty, spolu s nameranými časmi výpočtov a údajmi o veľkosti zabranej pamäte a výpočte smerodajnej odchýlky sú v tabuľkách 6.4 a 6.5.

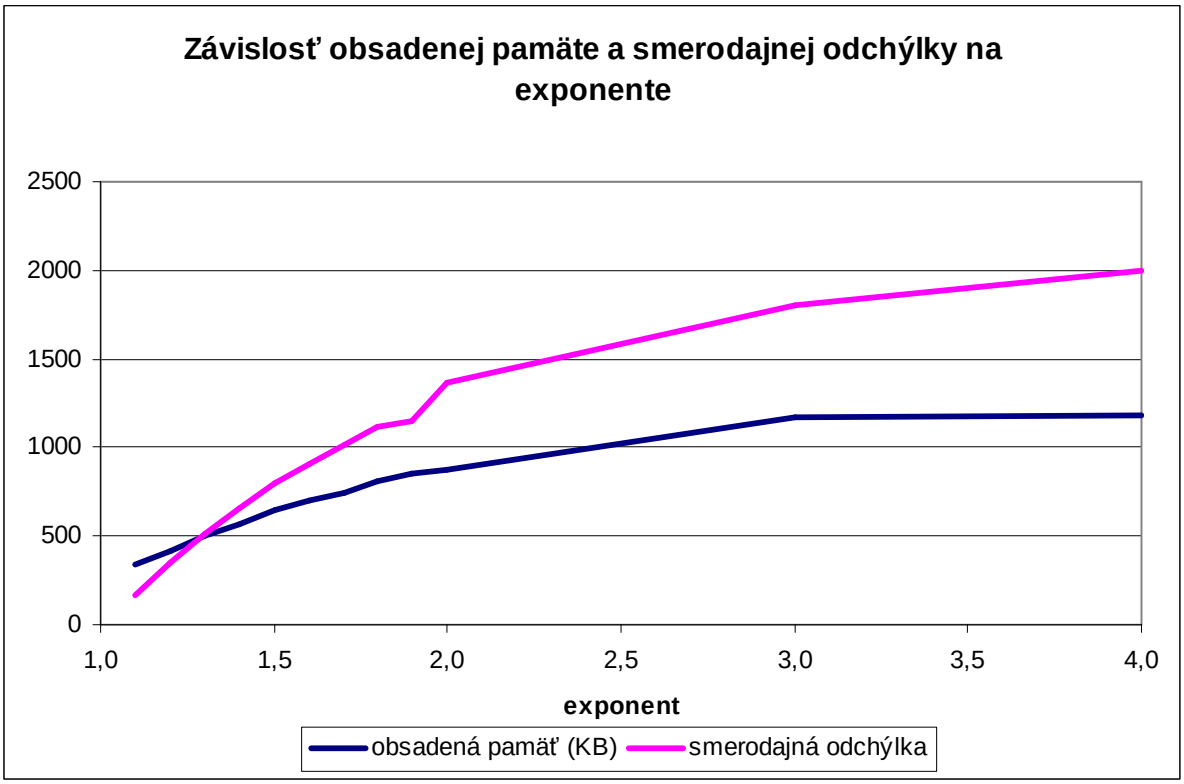
exponent		**4,0**	**3,0**	**2,0**	**1,9**	**1,8**	**1,7**
kandidátne 2-množiny v jednotlivých partíciách	Pp	4186	2775	378	2211	1035	703
	Pp-1	5544	5481	3808	3675	3430	3038
	Pp-2	1746	2475	3440	2760	2916	2814
	Pp-3	459	897	2104	1507	2072	2091
	Pp-4		307	1148	1022	1132	1365
	Pp-5			598	453	740	867
	Pp-6			305	307	303	598
	Pp-7			154		307	305
	Pp-8						154
obsadená pamäť (KB)		1176,896	1166,640	869,776	856,336	805,656	741,992
smerodajná odchýlka		1994,16	1804,31	1364,32	1151,25	1109,16	1015,25
čas výpočtu (s)		50,562	52,593	52,203	49,109	49,125	49,11

Tabuľka 6.4: Získané hodnoty metódy exponenciálneho rozdelenia partícií pre exponenty od 4 do 1,7

exponent	**1,6**	**1,5**	**1,4**	**1,3**	**1,2**	**1,1**
obsadená pamäť (KB)	694,008	644,744	572,792	501,560	415,736	337,272
smerodajná odchýlka	904,02	801,38	650,52	514,25	349,00	165,02
čas výpočtu (s)	49,391	49,625	49,297	50,219	50,563	53,422

Tabuľka 6.5: Získané hodnoty metódy exponenciálneho rozdelenia partícií pre exponenty od 1,6 do 1,1

Ako môžeme vypozorovať z tabuliek ako aj z priebehu kriviek grafu 6.2, so znižujúcim sa exponentom sa znižuje aj pamäťová náročnosť algoritmu. Napríklad pre exponent 1,1 je smerodajná odchýlka 162,02 a obsadená pamäť najnižšia, a to 337,272KB avšak oproti prechádzajúcim prípadom sa čas výpočtu predĺžil o 5,7% na 53,422s.



Graf 6.2: Závislosť obsadenej pamäte a smerodajnej odchýlky na voľbe exponenentu

Napriek tomu, že metóda exponenciálneho rozdelenia partícií vykazuje o trochu lepšiu efektivitu výpočtu, pri pohľade na smerodajnú odchýlku vidíme pretrvávajúce veľké rozdiely v počte vygenerovaných kandidátnych 2-množín v jednotlivých partíciách.

6.2.4 Rozdelenie zamerané na počty kandidátnych 2-množín

Cieľom tejto metódy je rozdelenie frekventovanej 1-množiny do partícií tak, aby ich pridávaním vznikali v každom kroku rovnaké, resp. veľmi podobné počty kandidátnych 2-množín. Za týmto prístupom stojí predpoklad zníženia smerodajnej odchýlky, a tým aj zníženie pamäťovej náročnosti Multipass algoritmu.

Táto metóda má dve možné modifikácie.

a) snaha o striktné dodržanie prednastaveného počtu partícií (striktný variant)
b) počet partícií nemusí byť dodržaný (benevolentný variant)

Proces výpočtu oboch metód delenia na partície opisuje nasledujúci pseudo-algoritmus:

```
stavnoveny_pocet_particii = ?? (prednastavená hodnota)
L1 = frekventovana 1-množina;
p = stavnoveny_pocet_particii;
celkovy_pocet_kandid_2mnozin = ((pocet_frekv_1mnozin – 1) ^ 2 +
    (pocet_frekv_1mnozin – 1) – mnozin) / 2;
```

```
pocet_kandid_2mnozin_v_kazdej_particii = celkovy_pocet_kandid_2mnozin /
     stavnoveny_pocet_particii;
D = (int) (1 - 4 * 1 * -2 * nrOf2ItmesIn1Partition);
x1 = Math.ceil((-1 + Math.sqrt(D)) / 2);//v x1 počet prvkov v partícii Pp;
limit_kandid_2mnozin_pre_1_particiu = (x1^2 + x1) / 2;

while (nie je spracovana posledna partícia P1) {
     vypocet frekventovanych mnozin v particii Pp;

     pocet_novych_prvkov = limit_kandid_2mnozin_pre_1_particiu  /
     pocet_uz_spracovanych_prvkov;
     if (pocet_novych_prvkov >= 2) {
          pocet_2mnozin_iba_z_novej_particie = ((pocet_novych_prvkov - 1)^2
               + (pocet_novych_prvkov – 1)) / 2;
          pocet_novych_prvkov = (limit_kandid_2mnozin_pre_1_particiu -
               pocet_2mnozin_iba_z_novej_particie) /
               pocet_uz_spracovanych_prvkov;
          Pp = pridaj nove prvky;
     } else {
          Pp = pridaj nove prvky;
     }
     pocet_particii++;

     // !!! rozdelenie algoritmu podla modifikacii !!!
     if (modifikacia == ’a’) {
          if (pocet_particii == stavnoveny_pocet_particii) {
               do Pp pridaj vsetky zvysne prvky z frekv_1mnoziny;
          }
     } else if (modifikacia == ’b’) {
          if  (p<0) {
               inkrementuj cislovanie particii;
          }
     }
     p--;
}
```

Popis pseudokódu:

Pri výpočte frekventovanej 1-množiny (L1) je zistený počet tejto množiny. Podľa vzorca 1 je z tohto údaju získaný počet kandidátnych 2-množín. Algoritmus je zameraný postupné pridávanie prvkov z množiny L1 tak, aby počet novovzniknutých kandidátnych 2-množín bol v každom kroku približne rovnaký. Na začiatku sa teda vypočíta odhadovaný počet kandidátnych 2-množín ktoré

vzniknú pri spracovaní každej z partícií. Z koreňa kvadratickej rovnice je následne získaný počet prvkov v partícii Pp a podľa vzorca 6.1 aj počet kandidátnych 2-množín, ktoré vzniknú pri spracovaní partície Pp. Potom dôjde k procesu získavania kandidátnych a následne frekventovaných množín v rámci danej partície. Po tomto kroku je potrebné spracovať ďalšiu partíciu a musí teda dôjsť k výpočtu počtu prvkov v ďalšej partícii (Pp-1). Vzhľadom k tomu, že kandidátne 2-množiny z novej partície (Pp-1) vzniknú kartézskym súčinom prvkov z už spracovanej Pp partície a novej (Pp-1) partície, počet prvkov sa určí podelením počtu kandidátnych 2-množín z partície Pp počtom už spracovaných prvkov. Tento výpočet však nezohľadňuje kandidátne 2-množiny, ktoré vzniknú kombináciou prvkov v rámci novej partície (Pp-1). A teda, pokiaľ je počet novo pridávaných prvkov väčší ako 2, dôjde k spresneniu výpočtu tak, že od počtu kandidátnych 2-množín z partície Pp sa odpočíta počet kandidátnych 2-množín vzniknutých z prvkov v rámci novej partície a až následne je výsledok tohto rozdielu podelený počtom už spracovaných prvkov. Tým sa zabráni vygenerovaniu priveľkého počtu kandidátnych 2-množín z nových partícií. V nasledujúcom kroku dôjde k rozvetveniu algoritmu podľa toho, aká modifikácia bola zvolená:

a) striktné dodržanie počtu partícií - pokiaľ nie je dovŕšený predpísaný počet partícií, nasleduje výpočet kandidátnych a následne aj frekventovaných množín a opäť výpočet počtu nových prvkov z ďalšej partície. Keď pri výpočtoch dôjde k naplneniu predpísaného počtu partícií, do poslednej partície sa pridajú všetky chýbajúce prvky z L1 bez ohľadu na to, koľko kandidátnych 2-množín z nich bude vyprodukovaných.

Už pri prvotných testoch s týmto variantom je vidieť (z tabuľky 6.6), že dochádza k nepríjemnému javu, a to nahromadeniu kandidátnych 2-množín v poslednej partícii podobne ako tomu bolo v metóde lineárneho rozdelenia. Tým sa potlačuje výhoda výpočtu rovnomerného počtu kandidátnych 2-množín, ktorá vznikli pridaním iných partícií. Z tohto dôvodu sa v modifikácii b netrvá na dodržaní počtu stanovených partícií, pričom tento preddefinovaný počet slúži iba ako vodítko výpočtu.

definovaný počet partícií	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
skutočný počet partícií	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
počty kandidátnych 2-množín	5995	4005	3003	2415	2016	1711	1540	1326	1225	1128	1035
	5940	3498	2668	2145	1812	1529	1386	1159	1053	1017	918
		4432	2844	2343	1950	1611	1445	1256	1125	1095	973
			3420	2277	1972	1590	1515	1309	1157	1038	990
				2755	1980	1687	1495	1278	1218	1078	1034
					2205	1602	1530	1298	1125	1085	936
						2205	1529	1285	1225	1062	1017
							1495	1242	1188	1012	972
								1782	1124	1076	903
									1495	994	952
										1350	1001
											1204
obsadená pamäť (KB)	*1246,512*	*978,032*	*803,840*	*694,784*	*599,640*	*600,025*	*492,200*	*527,496*	*480,256*	*455,472*	*431,680*
smerodajná odchýlka	*27,50*	*381,77*	*278,35*	*204,36*	*115,91*	*211,80*	*48,87*	*167,61*	*113,49*	*92,08*	*75,32*
čas výpočtu (s)	*48,688*	*48,672*	*48,906*	*48,938*	*48,938*	*48,672*	*48,234*	*48,047*	*48,031*	*48,140*	*48,328*

Tabuľka 6.6: Získané hodnoty variantu striktného dodržania počtu partícií metódy zameranej na počet kandidátnych 2-množín pre definovaný počet partícií 2 až 12

b) dodržanie predpísaného počtu partícií nie je nevyhnutné (benevolentný variant) - pokiaľ je skutočný počet partícií väčší ako prednastavený, partície sú prečíslované tak, aby nedošlo k číslovaniu partícií zápornými číslami. Do počtu partícií nie je počas výpočtu nijako zasahované, a teda sa stáva, že z poslednej partície je vytvorený omnoho menší počet kandidátnych 2-množín než z predchádzajúcich partícií. Ako si môžeme všimnúť v tabuľkách 6.7 a 6.8 ani počty definovaných a skutočných partícií nemusia súhlasiť.

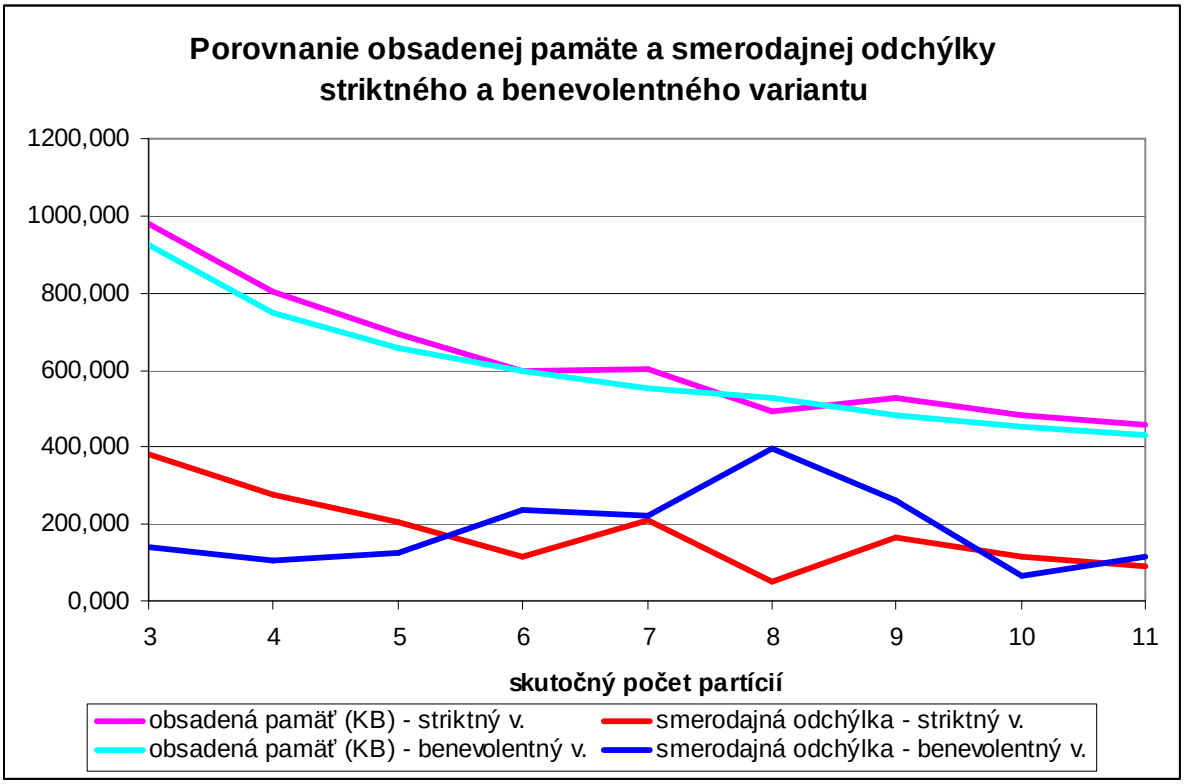
definovaný počet partícií	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
skutočný počet partícií	**3**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**10**	**11**
počty kandidátnych 2-množín	6105	4095	3081	2485	2080	1770	1596	1378	1275	1176	1081
	5523	3780	2805	2268	1925	1633	1485	1250	1210	1102	999
	307	4060	3025	2507	2100	1748	1575	1377	1343	1208	1080
			3024	2470	2151	1870	1672	1455	1323	1170	1118
				2205	2184	1890	1673	1443	1290	1230	1182
					1495	1820	1729	1482	1309	1254	1095
						1204	1746	1485	1295	1245	1195
							459	1455	1395	1206	1161
								610	1341	1287	1100
									154	1057	1164
											760
obsadená pamäť (KB)	*1265,008*	*921,936*	*746,072*	*659,200*	*599,328*	*551,400*	*525,504*	*481,624*	*466,456*	*450,560*	*433,448*
smerodajná odchýlka	2606,88	140,97	105,75	125,04	235,59	219,39	398,54	262,54	349,56	66,80	115,99
čas výpočtu (s)	*50,109*	*51,906*	*50,328*	*50,344*	*51,125*	*50,563*	*50,391*	*49,812*	*49,084*	*49,759*	*49,875*

Tabuľka 6.7: Získané hodnoty variantu benevolentného dodržania počtu partícií metódy zameranej na počet kandidátnych 2-množín pre definovaný počet partícií 2 až 12

definovaný počet partícií	**25**	**50**	**75**	**90**	**100**	**150**	**200**	**250**
skutočný počet partícií	**21**	**45**	**79**	**94**	**100**	**113**	**123**	**128**
obsadená pamäť (KB)	*340,760*	*303,720*	*279,704*	*271,104*	*272,008*	*274,456*	*276,056*	*276,856*
čas výpočtu (s)	*49,375*	*52,937*	*58,078*	*60,454*	*61,156*	*63,235*	*65,023*	*65,594*

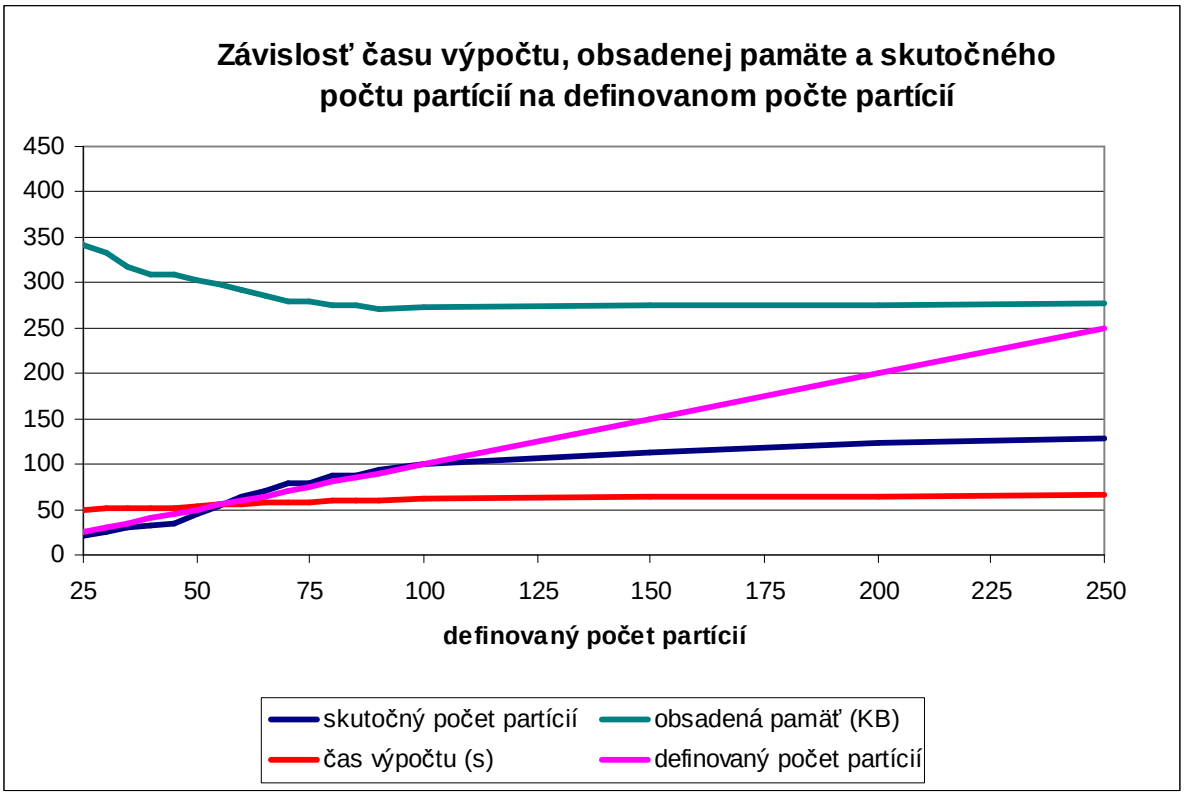
Tabuľka 6.8: Získané hodnoty variantu benevolentného dodržania počtu partícií metódy zameranej na počet kandidátnych 2-množín pre definovaný počet partícií 25 až 250

Napriek týmto, na prvý pohľad nedostatkom, pri porovnaní obsadenej pamäte pre rovnaké počty skutočných partícií (3-11) tohto a predchádzajúceho variantu je tento variant efektívnejší, čo nám dokazuje aj graf 6.3.



Graf 6.3: Porovnanie efektivity striktného a benevolentného variantu

Najvhodnejšia voľba počtu partícií pre tento prípad bola hodnota 90, kedy algoritmu zaberal najmenšie množstvo pamäte, a to 271,104KB. Tento fakt dokazuje aj graf 6.4 kde okrem tohto údaju je zachytená aj závislosť času výpočtu a skutočného počtu partícií na definovanom počte partícií.



Graf 6.4: Celková efektivita benevolentného variantu

Ako je možné si všimnúť, skutočný počet partícií nemá priamy súvis s definovaným počtom partícií. To nás privádza k myšlienke, zmeniť princíp algoritmu tak, že nebude nutné zadávať definovaný počet partícií, iba pomer počtu kandidátnych 2-množín, ktoré vzniknú z prvkov každej partície k celkovému počtu všetkých kandidátnych 2-množín, tak ako je to v nasledujúcej metóde.

6.2.5 Pomerové rozdelenie zamerané na počty kandidátnych 2-množín

V tejto metóde je postup veľmi podobný s metódou predchádzajúcou, avšak s tým rozdielom, že sa nesnažíme o určenie počtu partícií, iba o vyjadrenie počtu kandidátnych 2-množín v partícii Pp, ktoré vzniknú z prednastaveného koeficientu. Ostatné partície sú volené tak, aby pridaním každej z nich vznikali podobné počty kandidátnych 2-množín, podobne ako tomu bolo v predchádzajúcej metóde. Postup výpočtu zachytáva nasledujúci pseudokód:

```
L1 = frekventovana 1-množina;
promile = 0.001 * ??(prednastavená hodnota)
celkovy_pocet_kandid_2mnozin = ((pocet_frekv_1mnozin – 1) ^ 2 +
    (pocet_frekv_1mnozin – 1) – mnozin) / 2;
pocet_kandid_2mnozin_v_kazdej_particii = celkovy_pocet_kandid_2mnozin *
    promile;
```

```
D = (int) (1 - 4 * 1 * -2 * nrOf2ItmesIn1Partition);
x1 = Math.ceil((-1 + Math.sqrt(D)) / 2);//v x1 počet prvkov v partícii Pp;
x1++; //lepsie vysledky
limit_kandid_2mnozin_pre_1_particiu = (int)
pocet_kandid_2mnozin_v_kazdej_particii;

while (nie je spracovana posledna partícia) {
     vypocet frekventovanych mnozin v aktualnej particii;

     pocet_novych_prvkov = limit_kandid_2mnozin_pre_1_particiu  /
     pocet_uz_spracovanych_prvkov;
     if (pocet_novych_prvkov >= 2) {
          pocet_2mnozin_iba_z_novej_particie = ((pocet_novych_prvkov - 1)^2
               + (pocet_novych_prvkov – 1)) / 2;
          pocet_novych_prvkov = (pocet_kandid_2mnozin_v_kazdej_particii -
               pocet_2mnozin_iba_z_novej_particie) /
               pocet_uz_spracovanych_prvkov;
          pridaj novú partíciu;
     } else {
          pridaj novú partíciu;
     }
}
```

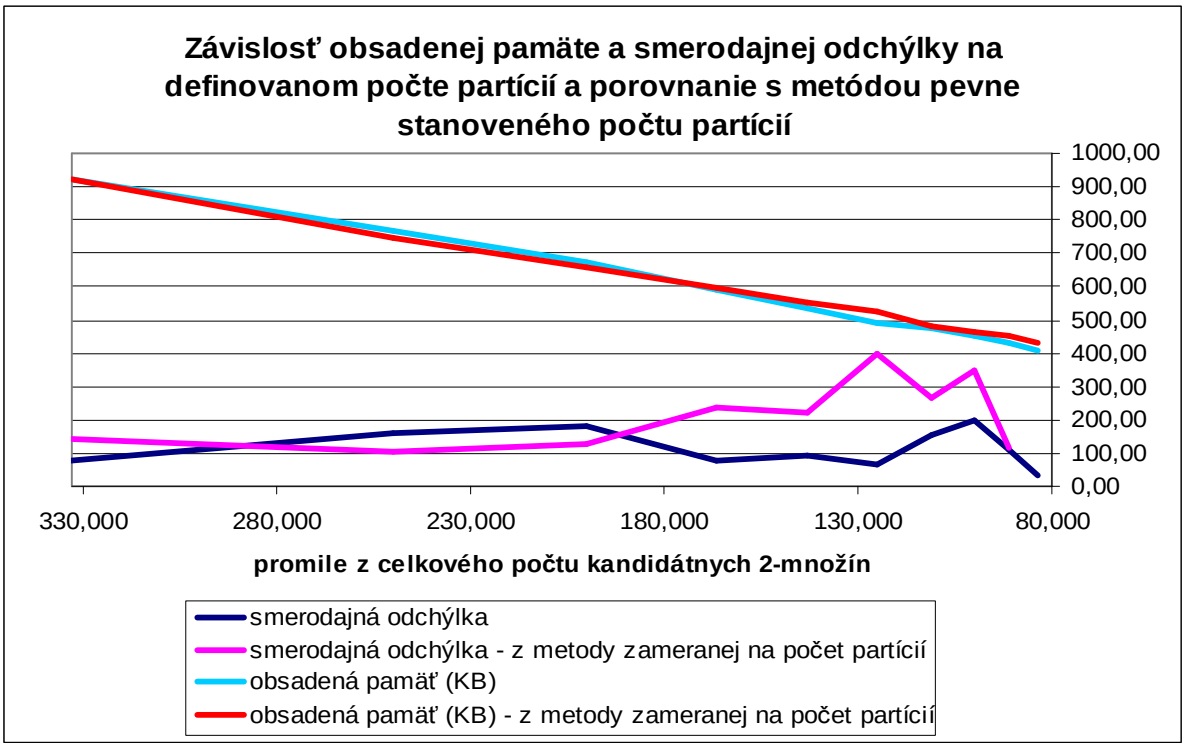
Algoritmus vychádza z predchádzajúcej metódy. Najväčší rozdiel spočíva v začiatku algoritmu, kde nestanovujeme definovaný počet partícií, ale pomerové vyjadrenie počtu kandidátnych 2-množín k celkovo vygenerovanému počtu kandidátnych 2-množín v rámci celej frekventovanej 1-množiny (L1). Počet prvkov (x1) v partícii Pp je opäť vypočítaný z kvadratickej rovnice, avšak v tomto prípade dôjde k ich navýšeniu o 1 prvok, z dôvodu vykazovania lepších výsledkov, kedy v poslednej partícii nezostáva zvyškový počet množín z frekventovanej 1-množiny, pretože tieto sú rovnomerne rozložené do zvyšných partícií.

Limit kandidátnych 2-množín pre partíciu Pp je stanovený jednoduchým zaokrúhlením smerom dolu predpočítanej premennej počtu kandidátnych 2-množín v každej partícii. Ďalšia časť výpočtu je prakticky totožná s benevolentným variantom predchádzajúcej metódy.

Tento na prvý pohľad malý rozdiel má za následok zvýšenie presnosti výpočtu, a tým zníženie potrebnej pamäte v porovnaní s metódou zameranou na stanovený počet partícií. Porovnanie pamäťovej náročnosti tejto metódy s predchádzajúcou zachytáva nasledujúca tabuľka 6.8 a graf 6.5.

promile	**333,333**	**250,000**	**200,000**	**166,667**	**142,857**	**125,000**	**111,111**	**100,000**	**90,909**	**83,333**
počet partícií	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
kandid. 2-mnoziny	3916	2926	2346	1891	1653	1431	1275	1176	1035	946
	4085	3069	2407	1937	1750	1495	1353	1239	1110	1007
	3934	3185	2507	2058	1748	1539	1377	1240	1176	973
		2755	2610	2115	1752	1530	1455	1295	1144	990
			2065	2010	1743	1508	1443	1266	1100	1034
				1924	1794	1542	1353	1287	1105	1045
					1495	1540	1335	1275	1080	1026
						1350	1435	1233	1161	980
							909	1314	1100	1044
								610	1164	966
									760	1015
										909
smerodajná odchýlka	*75,78*	*160,80*	*184,33*	*79,17*	*94,30*	*63,84*	*157,35*	*198,00*	*110,13*	*32,06*
smerodajná odchýlka - z metódy zameranej na počet partícií (benevolentný variant)	*140,97*	*105,75*	*125,04*	*235,59*	*219,39*	*398,54*	*262,54*	*349,56*	*115,99*	
čas výpočtu (s)	*50,328*	*49,156*	*49,281*	*49,312*	*50,375*	*49,016*	*48,547*	*48,500*	*49,719*	*49,671*
obsadená pamäť (KB)	*925,056*	*769,68*	*674,28*	*588,872*	*534,352*	*494,424*	*475,72*	*453,288*	*429,816*	*410,568*
obsadená pamäť (KB) - z metódy zameranej na počet partícií (benevolentný variant)	*921,936*	*746,072*	*659,200*	*599,328*	*551,4*	*525,504*	*481,624*	*466,456*	*450,56*	*433,448*

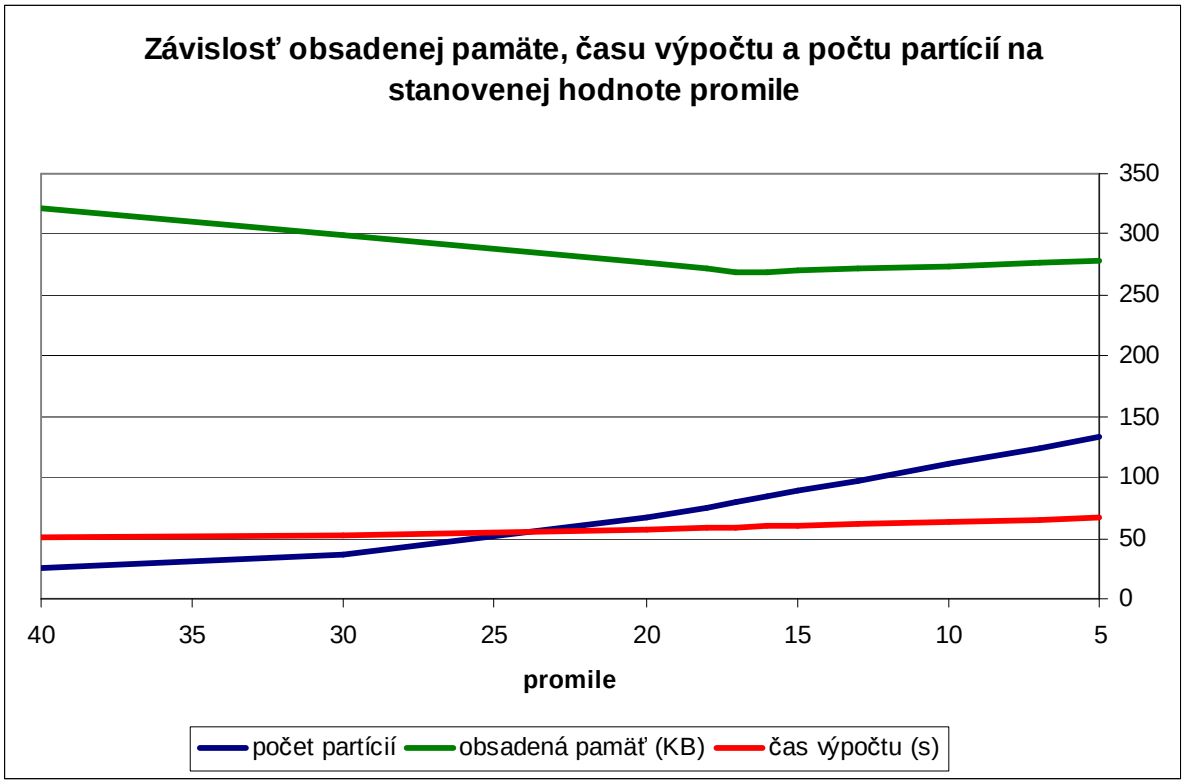
Tabuľka 6.9: Porovnanie efektivity metódy pomerového rozdelenia frekventovaných 1-množín na partície s metódou rozdelenia na definovaný počet partícií (benevolentný variant)



Graf 6.5: Porovnanie získaných hodnôt metódy pomerového rozdelenia na partície a benevolentného variantu metódy rozdelenia na definovaný počet partícií

Z grafu môžeme vyčítať, že rozdelenie prvkov z frekventovanej množiny do partícií je v tejto metóde rovnomernejšie než v metóde predchádzajúcej. Z toho vyplýva aj o trochu menšia pamäťová náročnosť.

Ako najvýhodnejšia voľba hodnoty promile bola hodnota 17 promile viď. nasledujúci graf 6.5. Pri tejto hodnote najväčší kus vyhradenej pamäte bol 268,712KB. Ďalším znižovaním hodnoty promile už nedochádza k zlepšeniu pamäťovej náročnosti, na druhú stranu však dochádza k zhoršovaniu časovej náročnosti.



Graf 6.6: Závislosť obsadenej pamäte, času výpočtu a počtu partícií na stanovenej hodnote promile

Pre inú kolekciu dát pozostávajúcu z 298 frekventovaných 1-množín, z ktorej je možné vytvoriť 44253 kandidátnych 2-množín viedla voľba 9 promile k najlepšiemu výsledku čo sa týka pamäťovej náročnosti (viď. tabuľka 6.9). Celkovo sa teda dá usúdiť, že voľba hodnoty promile od 5 promile do 20 promile je najefektívnejšia z hľadiska pamäťovej náročnosti.

promile	**25**	**17**	**10**	**9**	**8**	**7**	**5**
obsadená pamäť (KB)	503,040	445,056	377,752	371,768	371,773	374,744	380,512
čas výpočtu (s)	117,093	120,390	131,125	133,578	136,172	138,938	143,875

Tabuľka 6.10: Závislosť obsadenej pamäte na voľbe hodnoty promile pre väčší počet frekventovaných 1-množín

6.2.6 Zhrnutie prístupov rozdelenia partícií

Táto kapitola bola venovaná rôznym prístupom rozdeľovania frekventovanej 1-množiny na partície. Boli zhodnotené prínosy a nedostatky jednotlivých metód a nakoniec bola odporučená metóda pomerového vyjadrenia počtu kandidátnych 2-množín v jednotlivých partíciách k celkovému počtu kandidátnych 2-množín. Pri tejto metóde dochádza k rovnomernému rozdeľovaniu frekventovanej 1-množiny na partície tak, aby bol počet kandidátnych 2-množín vzniknutých z jednotlivých partícií približne rovnaký. To má za následok relatívne nízke hodnoty smerodajnej odchýlky kandidátnych 2-množín jednotlivých partícií, a najmä s tým spojené veľmi dobré výsledky z hľadiska pamäťovej náročnosti. Z časového hľadiska sa dá povedať, že medzi metódami nebol výrazný rozdiel, až na metódu inkrementálneho rozdelenia, kedy bol čas výpočtu približne 38% vyšší ako v ostatných metódach.

Bez ohľadu na voľbu prístupu rozdelenia , všetky metódy dosahovali niekoľkonásobne (10 až 16 násobne) nižšie množstvá obsadenej pamäte oproti metóde Apriori. V tom je najväčší prínos metódy Multipass-Apriori ako takej.

6.3 Vplyv usporiadania na efektivitu výpočtu

Jednou z možností, ako sa dá ovplyvniť efektivita výpočtu je zvolenie usporiadania v rámci kandidátnych respektíve frekventovaných množín. Zvolené usporiadanie ovplyvní usporiadanie vo všetkých kandidátnych aj frekventovaných k-množinách v celej aplikácii. Na výber máme jedno zo štyroch možností usporiadania:

1. od najnižšej podpory k najvyššej (napr. jablko:3, keksy:3, ryba:5, konzerva:6, atď.)
2. od najvyššej podpory k najnižšej (napr. maslo:7, konzerva:6, maso:6, atď.)
3. abecedné usporiadanie od A do Z (napr. jablko:3, keksy:3, konzerva:6, atď.)
4. abecedné usporiadanie od Z do A (napr. syr:6, ryba:5, rozok:6, atď.)

Pri voľbe podpory 3% dopadli testy časovej a pamäťovej náročnosti pre jednotlivé typy usporiadania nasledovne(pre porovnanie sú v tabuľke uvedené aj namerané hodnoty pre metódu Apriori):

číslo metódy usporiadania	čas výpočtu	využitá pamäť
1	5min 44,734s	1MB 688KB 480B
2	5min 45,823	1MB 680KB 174B
3	5min 43,891s	1MB 715KB 688B
4	5min 51,862s	1MB 749KB 564B
metóda Apriori	5min 42,219s	27MB 804KB 400B

Tabuľka 6.11: porovnanie časovej a pamäťovej efektivity jednotlivých prístupov usporiadania a metódy Apriori pre podporu 3%

Z výsledkov môžeme usúdiť, že metóda abecedného usporiadania od Z do A (č. 4) má najhoršie výsledky čo sa týka pamäťovej aj časovej efektivity. Medzi ostatnými metódami nie je rozdiel veľký, avšak ako najvýhodnejšia sa javí metóda abecedného usporiadania od A do Z (č. 3) nasledovaná metódou usporiadania od najnižšej podpory k najvyššej (č. 1). Metóda usporiadania od najvyššej podpory k najnižšej (č. 2) bola vyhodnotená ako tretia najlepšia. Metóda Apriori vyšla z testov paradoxne ako najrýchlejšia, avšak v jej neprospech hovorí množstvo využitej pamäte, ktoré je 16-násobne väčšie než pri ktorejkoľvek z metód Multipass.

Na určenie najefektívnejšej voľby usporiadanie je nutné vykonať ďalšie testy s nižšou podporou. Preto bola podpora znížená na 2% a testy boli vykonané na metódach číslo 1, 2, 3 a Apriori. Metóda 4 bola z ďalšieho testovania vylúčená z dôvodu najhoršej efektivity už po prvom teste. Výsledky testov efektivity pre podporu 2% zachytáva nasledujúca tabuľka:

číslo metódy usporiadania	čas výpočtu	využitá pamäť
1	14min 15,266s	4MB 545KB 408B
2	14min 32,797s	4MB 937KB 40B
3	13min 36,859s	4MB 266KB 672B
Apriori	58min 46,375s	61MB 884KB 8B

Tabuľka 6.12: porovnanie časovej a pamäťovej efektivity jednotlivých prístupov usporiadania a metódy Apriori pre podporu 2%

V prvom rade treba poznamenať, že metóda Apriori bola približne 4x pomalšia než všetky ostatné metódy. Jej dobrý časový výsledok z testu s podporou 3% sa nezopakoval, navyše podľa predpokladu bola metóda Apriori približne 14x pamäťovo náročnejšia. Ako najvhodnejšia z metód sa javí metóda abecedného usporiadania od A do Z, ktorá aj v tomto teste bola najefektívnejšia. Táto metóda bude výhradne využívaná pri získavaní asociačných pravidiel z dokumentov. Z metód usporiadania podľa podpory bola efektívnejšia metóda usporiadania od najnižšej podpory k najvyššej, avšak ako už bolo spomenuté, nie je taká efektívna, ako metóda abecedného usporiadania.

7 Aplikácia

Aplikácia je naprogramovaná v programovacom jazyku Java 2 Standard Edition 1.5 a naplno využíva vylepšenia oproti minulým verziám, hlavne generické (parametrizované) typy a konštrukcie s tým spojené. Jedná sa o konzolovú aplikáciu spúšťanú z príkazového riadka. Vývojovým prostredím použitým na vývoj aplikácie bolo univerzálne prostredie Eclipse 3.2.

7.1 Prehľad tried

Celá aplikácia pozostáva z celkovo 11 tried, ktoré majú nasledovnú funkciu:

Main.java - jedná sa o hlavnú triedu aplikácie. Na začiatku je vypísaná úvodná obrazovka a získaný vstupný parameter, ktorým je súbor s konfiguračným nastavením, na základe ktorého sa vytvorí inštancia triedy Configuration. Podľa nastavenia statických premenných v triede Configuration, je vytvorená trieda Preprocess alebo Mining, v závere je vždy vytvorená inštancia triedy AssociationRules.

Configuration.java - trieda slúžiaca na spracovanie XML dokumentu s konfiguráciou, ktorá bola zadaná ako parameter vstupu. Najdôležitejšou metódou je metóda ParseXML(), ktorá ako názov napovedá rozparsruje konfiguračný súbor, ktorý je vo formáte XML a podľa jeho obsahu nastaví statické vlastnosti, ktorými sa riadi celá aplikácia.

Preprocess.java - táto trieda slúži na predspracovanie kolekcia dát Reuters-21578. Najdôležitejšia metóda je preprocessSGML() ktorá robí funkciu parsera SGML dokumentu z ktoré vyberá tokeny nachádzajúce sa medzi dôležitými značkami, ktoré spracováva. Na každý token je volaná metóda advancePreprocess(String token) v ktorej sú uplatnené pravidlá predspracovania. Výstupom sú dva súbory - v jednom sú transakcie vzniknuté z obsahov dokumentov a v druhom popisy týchto dokumentov (viď. kapitoly 3.3).

Stemmer.java - jedná sa o upravenú verziu implementácie originálneho Porterovho stemmovacieho algoritmu. Metódy využívajúce najmä neobjektové vlastnosti Javy sú upravené tak, aby algoritmus nebol pri odstraňovaní koncoviek tak agresívny. Snahou je o pokrytie čo najväčšej škály možných typov slov a výnimiek.

Mining.java - podľa nastavených konfiguračných nastavení sa v tejto rozhoduje, či proces dolovania bude vykonávaný metódou Apriori, alebo Multiapass-Apriori. Je tu vytvorená inštancia triedy Apriori a podľa potreby sú volané metódy tejto triedy. Takisto sú tu vytvorené inštancie tried StopWatch a MemoryUsage.

Apriori.java - trieda, v ktorej je implementácia metódy získavania frekventovaných Apriori. Metódy tejto triedy sú naprogramované tak, že ich volaním je možné túto triedu použiť ako Multipass-Apriori.

StopWatch.java - je trieda ako názov napovedá slúžiaca na stopovanie času. Získa čas, ktorý ubehol medzi zavolaním metód tejto triedy start() a stop(). Jej využitie je pri získavaní času trvania výpočtu metód Apriori a Multipass-Apriori.

MemoryUsage.java - funkciou tejto prevzatej metódy je získanie rozdielu využitej pamäte pri volaní metód start() a end(). Údaj v bytoch je pomocou metódy memorySizeToString(long l) prevedený do užívateľsky prívetivej podoby v MB, KB, a B.

AssociationRules.java - trieda zodpovedná za vytvorenie asociačných pravidiel z frekventovaných množín. Rekurzívnym volaním metódy allSubsets() sú postupne vytvorené asociačné pravidlá a na výstup vypísané tie, ktoré spĺňajú podmienku minimálnej spoľahlivosti.

Itemset.java - veľmi významná trieda implementujúca rozhranie Comparable. Inštancie tejto triedy sú položkami každej kandidátnej a frekventovanej n-množiny. Obsahuje niekoľko typov konštruktorov, slúžiacich na vytvorenie k+1 množín z daných k-množín. Takisto nutne obsahuje implementáciu metódy compareTo() pomocou ktorej sú prvky v k-množinách usporiadané, a to buď podľa podpory alebo abecedne.

ParseError.java - trieda odvodená od triedy Exception slúžiaca na výpis chybového hlásenia, ktoré vzniklo pri spracovávaní súboru s konfiguráciou.

7.2 Štruktúra konfiguračného súboru

Jediným a povinným parametrom aplikácie je názov súboru vo formáte XML. V ňom je uvedená konfigurácia podľa ktorej je riadený celý proces predspracovania a získavania asociačných pravidiel. Súbor má pevnú štruktúru, a v nasledujúcej časti uvediem konkrétny príklad konfigurácie:

```
<?xml version="1.0"?>
<configuration>
      <verbosity>2</verbosity>
      <preprocess>
       <dopreprocess>0</dopreprocess>
       <stopwordsfile>stop_words.dat</stopwordsfile>
       <outputfile>output.txt</outputfile>
       <descriptionfile>output-dscr.txt</descriptionfile>
       <inputpreprocess>
            <file>reut2-000.sgm</file>
            <file>reut2-001.sgm</file>
       </inputpreprocess>
      </preprocess>
      <mining>
       <domining>1</domining>
```

```
        <support>3</support>
        <confidence>66%</confidence>
        <apriori>1</apriori>
        <multipassapriori>1</multipassapriori>
        <inputminingfile>test2.txt</inputminingfile>
      </mining>
  </configuration>
```

Na začiatku súboru je úvodný riadok, ktorý definuje, že sa jedná o súbor XML. Koreňovou značkou je <configuration> a ukončovacia </configuration>. Medzi týmito značkami sa nachádzajú všetky ostatné značky, ktoré môžu byť uvedené v ľubovoľnom poradí. Jedná sa o značky:

<verbosity> a </verbosity> - číslo uvedené medzi týmito značkami uvádza stupeň toho, koľko a aké informácie sa budú vypisované na výstup.

Hodnota 0 - na výstup sú vypísané jedine získané asociačné pravidlá.

Pri zadanej hodnote 2 sú vypísané všetky dostupné informácie. Je vypísaná hlavička aj s komentárom, aká časť výpočtu momentálne prebieha, sú vypísané frekventované množiny v hranatých zátvorkách aj s uvedenou početnosťou výskytu (podporou). Samozrejme sú vypísané asociačné pravidlá.

Zadaná hodnota 1 je kompromisom medzi obidvoma predchádzajúcimi voľbami. Rozdiel oproti hodnote 2 spočíva v tom, že frekventované množiny nemajú uvedenú podporu. To môže byť užitočné pri dodatočnom spracovávaní výsledkov, kedy tieto hodnoty početnosti výskytov nie sú potrebné.

<preprocess> a </preprocess> - sú značky obaľujúce celý rad značiek týkajúcich sa predspracovania. V nasledujúcej časti, tam kde by nemalo prísť k omylu budem z dôvodu väčšej čitateľnosti textu uvádzať iba uvádzacie značky. Z definície XML dokumentu vyplýva, že každá takáto značka musí obsahovať značku ukončovaciu (napr. pre uvedenú uvádzaciu značku <apriori> musí existovať značka </apriori>, ktorú však už v tomto texte neuvediem).

Dôležitou je značka <dopreprocess> s povolenými hodnotami 0 alebo 1, ktoré vyjadrujú, či sa má vykonávať proces predspracovania (hodnota 1) alebo nemá (hodnota 0).

Reťazec nasledujúci za značkou <stopwordsfile> udáva názov súboru obsahujúci nepotrebné slová (stop words).

<outputfile> - určuje názov výstupného súboru predspracovania, ktorý bude obsahovať transakcie s predspracovanými slovami, ktoré vznikli v tejto fáze.

<descriptionfile> - uvádza názov súboru obsahujúci popis dokumentov (transakcií) uvedených v predchádzajúcom súbore.

Medzi značkami <inputpreprocess> a </inputpreprocess> môže byť uvedených ľubovoľný počet dvojíc značiek <file> a </file> obsahujúcich vstupné súbory, ktoré majú byť predspracované.

Výsledkom predspracovania týchto súborov ako už bolo spomenuté bude jediný súbor, ktorého názov je uvedený medzi značkami <outputfile> a </outputfile>

<mining> a </mining> - sú značky v tejto pasáži tentokrát obaľujúce značky týkajúce sa dolovania.

Podobne ako v časti týkajúcej sa preprocesingu bola dôležitou značkou značka <dopreprocess> je tu podobnou značka <domining> - kde nasledujúce číselné hodnoty udávajú, či sa má vykonávať proces dolovania (hodnota 1) alebo nie (hodnota 0).

<support> - číselná hodnota uvedená za touto značkou môže udávať absolútnu podporu(napr. 3) alebo relatívnu podporu (číselná hodnota nasledovaná znakom %, napr. 2%).

<confidence> - nasledujúca celočíselná hodnota so znakom % udáva relatívnu spoľahlivosť (napr. 66%).

<apriori> - hodnota 1 nasledujúca za touto značkou označuje skutočnosť, že má byť na vyhľadávanie frekventovaných množín použitý algoritmus Apriori.

<multipassapriori> - podobne ako v predchádzajúcom prípade, hodnota 1 značí, že na vyhľadávanie frekventovaných množín použitý tentokrát algoritmus Multipass-Apriori.

<inputminingfile> - reťazec nasledujúci za touto značkou označuje názov súboru, ktorý bude použitý ako vstup pre dolovanie. Spravidla to býva výstup súboru predspracovania, teda súbor uvedený medzi značkami <outputfile> a </outputfile>.

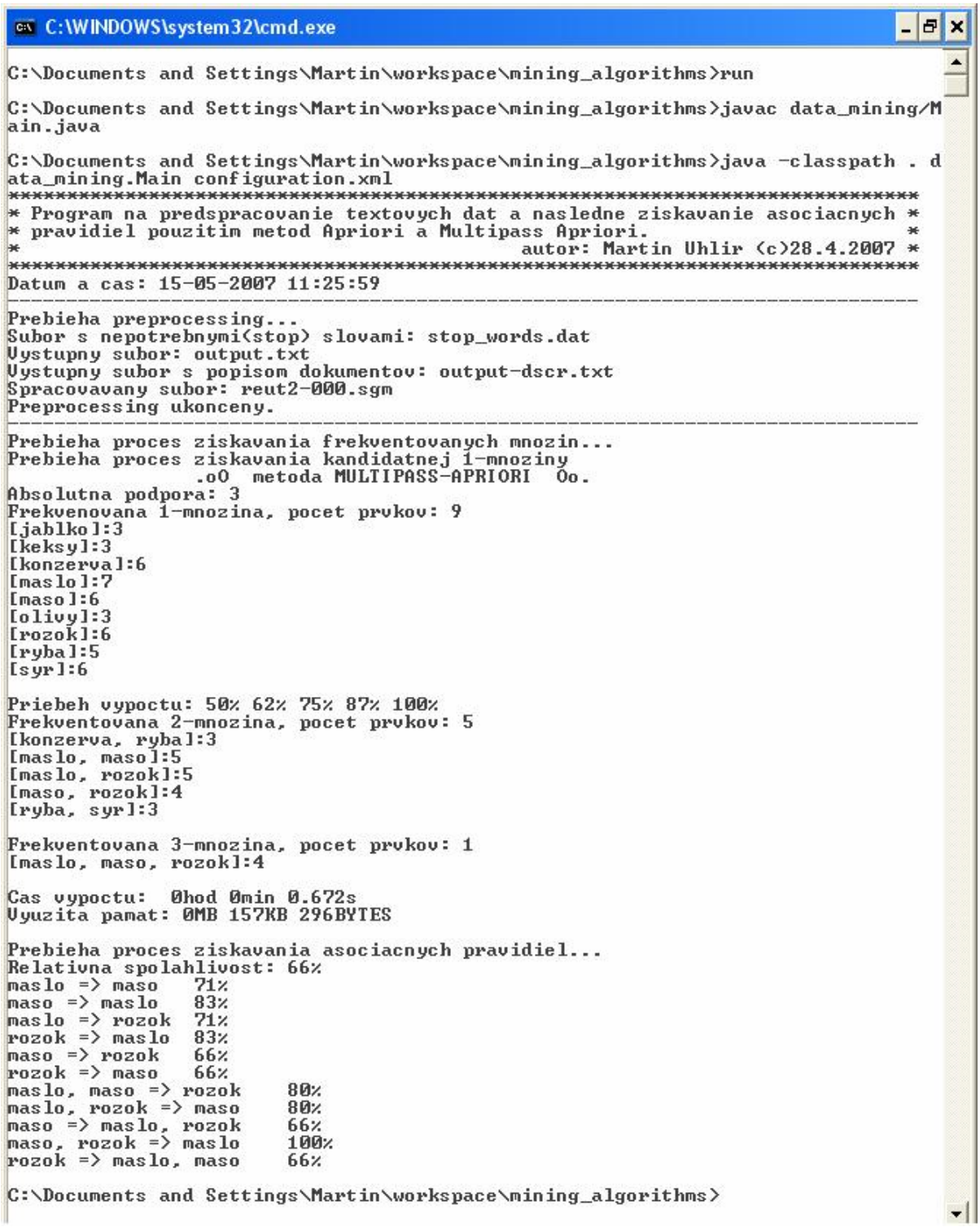
7.3 Vzhľad aplikácie

Ako už bolo spomenuté, jedná sa o konzolovú aplikáciu. Kompilácia sa vykonáva z príkazového riadku príkazom: `javac data_mining/Main.java`.

Po úspešnom skompilovaní je možné aplikáciu spustiť príkazom `java -classpath . data_mining.Main nazov_konfiguracneho_suboru`. Príklad spustenia: `java -classpath . data_mining.Main configuration.xml`.

Výhodou výstupu na štandardný výstup je možnosť presmerovania výstupu do súboru (napr. príkazom `java -classpath . data_mining.Main configuration.xml > association_rules.txt`). Výstupný súbor je možné modifikovať podľa potreby, prípadne asociačné pravidlá vložiť do tabuľkového procesoru (napr. Microsoft Excel alebo OpenOffice Calc) a tu ľubovoľným spôsobom usporiadať.

Súčasťou odovzdaných súborov je aj dávka run.bat, ktorá môže byť takisto použitá na kompiláciu a následné spustenie aplikácie.



Obr. 7.1 Ukážka výstupu aplikácie

8 Záver

V práci bola popísaná implementácia a využitie algoritmu Multipass-Apriori na získavanie asociačných pravidiel z textových dát. Jedná sa následníka algoritmu Apriori, ktorý je napríklad využívaný pri získavaní asociačných pravidiel z transakčných databáz. Textové databáze sa spravidla vyznačujú väčším počtom položiek (slov), a tak algoritmus Apriori nie je pre svoju pamäťovú náročnosť vhodný pre túto oblasť použitia. Ako bolo ukázané na praktických testoch, algoritmus Multipass-Apriori potrebuje k dosiahnutiu rovnakých výsledkov (získaniu frekventovaných množín) podstatne menšie množstvo pamäte. Využitím testov bol takisto určený najefektívnejší spôsob rozdelenia frekventovaných 1-množín na partície.

Veľmi významnú úlohu pri získavaní asociačných pravidiel z dát zohráva predspracovanie týchto dát. V aplikácii je využitých niekoľko spôsobov predspracovania, najmä však využitie slovníka nepotrebných slov a stemmovacieho algoritmu. Použitím užívateľsky meniteľného slovníka, ktorý obsahuje 127 slov sa zredukoval počet slov vo výstupnom súbore o 44% čím sa výpočet rapídne zrýchlil. Využitie slovníka prinieslo takisto zredukovanie počtu asociačných pravidiel najmä o také, ktoré nie sú prínosné z hľadiska získavania znalostí. Stemmer použitý v aplikácii vychádza z originálneho Porterovho stemmeru. Algoritmus bol značne prerobený tak, aby vyhovoval požiadavkám aplikácie. Zmeny sa týkali najmä drastického odstraňovania koncoviek, ktoré je rysom originálneho algoritmu. Veľmi veľa pravidiel bolo vytvorených ad-hoc spôsobom čo malo dopad na elegantnosť algoritmu, nemalo to však výrazný dopad na časovú efektivitu výpočtu a čo je podstatné, prinieslo to výrazné zlepšenie korektnosti stemmingu.

Výsledkom dolovania dát z textov (v tomto prípade kolekcie textov Reuters-21578) sú asociačné pravidlá slúžiace na získavanie znalostí z textových dát. Po dodatočných úpravách metódy predspracovania by bolo možné algoritmus použiť na získavania asociačných pravidiel takmer z ľubovoľného typu textových dát.

8.1 Ďalšie možnosti vývoja

Algoritmus Multipass-Apriori navzdory svojej pamäťovej efektivite neodstraňuje nutnosť generovania veľkého počtu kandidátnych množín. Táto vlastnosť je totiž daná princípom algoritmu Apriori, ktorý algoritmus Multipass-Apriori využíva. Zvýšenie časovej efektivity by sa dalo dosiahnuť využitím pokročilej hashovacej techniky rozdelenia položiek do korešpondujúcich košov. Takisto vyradenie takých transakcií, ktoré neobsahujú žiadne frekventované k-množiny, by mohlo viesť k zvýšeniu efektivity, pretože takéto transakcie nebudú obsahovať ani žiadne frekventované k+1-množiny.

Väčšie možnosti vývoja vidím vo vylepšení efektivity predspracovania. Časť predspracovania týkajúca sa parsrovania môže byť prerobená univerzálnejším spôsobom na širšiu škálu štruktúrovaných aj neštruktúrovaných dokumentov. Slovník nepotrebných slov by po vykonaní rozsiahlej sady testov mohol byť rozšírený, príp. redukovaný tak, aby vzniknuté asociačné pravidlá lepšie korešpondovali s očakávanými výsledkami v podobe asociačných pravidiel. K zvýšeniu korektnosti predspracovania by mohli prispieť slovníky názvov štátov, miest, mien, atď. tak, aby algoritmus ponechal počiatočné písmená týchto názvov veľkými písmenami.

Stemmovací algoritmus môže byť rozšírený o nové pravidlá, takisto je možnosť zredukovania počtu pravidiel vytvorením pravidiel komplexnejších z jednotlivých čiastočných pravidiel. Vo fáze predspracovania je možnosť použitia ďalších techník, ktoré by mohli viesť k zvýšeniu efektivity získavania znalostí z textových dát.

Literatúra

[1] Berka, P.: Dobývání znalostí z databází, 1. vydanie, Praha, Academia, 2003

[2] Han, J., Kamber, M.: Data Mining: Concepts and Techniques, Second Edition, Elsevier Inc., 2006, 770 p., ISBN 1-55860-901-3.

[3] Hearst, M.: What is Text Mining?, SIMS, UC Berkeley, October 17, 2003, Dokument dostupný na URL http://www.ischool.berkeley.edu/~hearst/text-mining.html (október 2006)

[4] Sedláček, P: Dolování v textech, [bakalářská práce], Fakulta informatiky, Masarykova univerzita, Brno 2003.

[5] Holt, J. D., Chung, S. M.: Efficient Mining of Association Rules In Text Databases, In: Proceedings of CIKM 99, Kansas City, USA, ACM, 1999, pp. 234-242

[6] Zendulka, J. a kol.: Získávání znalostí z databází (studijní opora), Fakulta informačních technologií, VUT Brno, 2006

[7] Popelínský, L. Dolování v datech, Fakulta informatiky, Masarykova univerzita, Brno 2004.

[8] Famili, A. et al.: Data Preprocessing and Intelligent Data Analysis, Intelligent data Analysis, 1. vydanie, Elsevier Science B.V., 1997, strany 3-23

[9] Žáčková, E.: Parciální syntaktická analýza (češtiny), [disertační práce], Fakulta informatiky, Masarykova univerzita, Brno 2002.

[10] Dávid D.: Data mining a predspracovanie dát, [diplomová práca], Fakulta informatiky, Masarykova univerzita, Brno 2007.

[11] Smékal L.: Získávání znalostí z textových dat, [ročníkový projekt], Fakulta informačních technologií, VUT Brno, 2005

[12] Stemming, poslední aktualizace 14. máj 2007, Wikipedia, Dokument dostupný na URL http://en.wikipedia.org/wiki/Stemming (máj 2007)

[13] Fuller M., Zobel J.: Conflation-based Comparison of Stemming algorithms, Department of Computer Science, RMIT, Australia, Dokument dostupný na URL www.cs.rmit.edu.au/~jz/fulltext/adcs98.ps (máj 2007)

[14] Reuters-21578 Test Collections, Dokument dostupný na URL http://www.daviddlewis.com/resources/testcollections/reuters21578/ (máj 2007)

[15] Porter M.: The Porter Stemming Algorithm, Január 2006, Dokument dostupný na URL http://www.tartarus.org/~martin/PorterStemmer/ (máj 2007)

[16] What is Porter Stemming?, Lancaster University, Dokument dostupný na URL http://www.comp.lancs.ac.uk/computing/research/stemming/general/porter.htm (máj 2007)

[17] Dudáš L., Prešovský K., Benikovský B.: Možnosti a použitie systémov GHSOM a LSI v spracovaní textových dokumentov D.D. Lewis, Reuters, [semestrálny projekt], FEI TU Košice, 2001

Zoznam príloh

Príloha 1: Úpravy Porterovho stemmovacieho algoritmu

Originálny algoritmus prešiel rozsiahlou úpravou. Bolo potrebné zmeniť všetky kroky algoritmu, dokonca jeden krok bol vynechaný. Úpravami sa dosiahlo menej agresívneho spôsobu odoberania koncoviek a vzniku slov, ktoré tzv. majú význam.

Ako už bolo spomenuté v kapitole 3.3.5, algoritmus pracuje v šiestich krokoch, ktoré budú v nasledujúcej časti popísané. Schematickým spôsobom sú opísané pravidlá úprav v jednotlivých krokoch. Pre lepšie pochopenie sú zavedené niektoré funkcie a značky:

→: značí proces zmeny slova, koncovky atď. Na pravej strane je originálne slovo, resp. koncovka slova a na ľavej výsledná zmena slova, resp. koncovky (napr. has → have = zmena slova „has“ na „have“, ends(ness) → „“ = odstránenie koncovky „ness“).

funkcia ends(reťazec): funkcia znamená, že pravidlo bude uplatnené pokiaľ slovo končí na reťazec uvedený v zátvorkách(napr.: ends(ied) → y = replied → reply)

funkcia m(): (= Measure = Veľkosť) určuje, koľkokrát sa v slove opakuje sled skupín spoluhlások a samohlások (slová „tr“, „ee“, „tree“, „y“, „by“ majú m() = 0; slová „trouble“, „oats“, „tree“, „ivy“ majú m() = 1; slová „troubles“, „private“, „oaten“, „orrery“ majú m() = 2; viac v [15]).

if (výraz): funkcia logickej podmienky, pokiaľ je výraz v zátvorkách pravdivý, je uplatnené dané pravidlo

add(reťazec): funkcia značiaca pridanie reťazca v zátvorkách na koniec slova, na ktoré je aplikované pravidlo

not(výraz): funkcia logickej negácie

&: logický operátor AND

Krok 1 – odstránenie plurálu, koncoviek -ed, -ing, atď.

Slovo končí na „s“: ends(sses) → ss
ends(ies) → y
ends(‘s) → „“
has → have
this, was, is, as, focus, taxes, surplus → bez zmeny
ends(ous) → bez zmeny //napr. famous
ends(shes) → sh //establishes → establish, crashes → crash

Slovo končí na „d“: ends(eed) → ee //agreed → agree
proceed → bez zmeny
ends(ied) → y //replied → reply, denied → deny, applied → apply

Slovo končí na „ed“: hundred, urged, continued → bez zmeny

ends(ed) → „“

Slovo končí na „ing“: something, sterling, peking → bez zmeny

ends(ing) → „“

Zmena koncového „at“ na „ate“: float, treat, coat → bez zmeny

ends(at) → ate

Zmena koncového „bl“ na „ble“: ends(bl) → ble

Zmena koncového „iz“ na „ize“: ends(iz) → ize

Odstránenie dvojitých spoluhlások: ends(nn) → n //plann → plan, beginn → begin, runn → run

ends(pp) → p //stripp → strip, dropp → drop, shipp → ship

ends(rr) → r //preferr → prefer, referr → refer

ends(tt) → t //submitt → submit, committ → commit

Krok 2 – tento krok bol z pôvodného algoritmu úplne odstránený. Funkciou tohto kroku bola zmena koncového „y“ na „i“. Vzhľadom k tomu, že táto zmena nie je potrebná, bol tento krok celkom vypustený.

Krok 3 – zmena dvojitých prípon na jednoduché. Teda -ization (= -ize plus -ation) sa zmení na -ize. Slová však musia vyhovovať funkcii m() > 1.

Zmena koncového „ational“ na „ate“: international → bez zmeny

if (m()>1) ends(ational) → ate

Zmena koncového „tional“ na „tion“: if (m()>1) tional → tion //additional → addition

Zmena koncového „izer“ na „ize“: if (m()>1) ends(izer) → ize //fertilizer → fertilize

Zmena koncového „bly“ na „ble“: if (m()>1) ends(bly) → ble //reasonably → reasonable

Zmena koncového „ally“ na „al“: if (m()>1) ends(ally) → al //originally → original

Zmena koncového „ently“ na „ent“: if (m()>1) ends(ently) → ent //recently → recent

Zmena koncového „ely“ na „e“: if (m()>1) ends(ely) → e //privately → private

Zmena koncového „ously“ na „ous“: if (m()>1) ends(ously) → ous //previously → previous

Zmena koncového „ization“ na „ize“: if (m()>1) ends(ization) → ize //organization → organize

Zmena koncového „ation“ na „ate“: destination, application, certification, implication,

communication, information, organisation → bez zmeny

if (m()>1) ends(ation) → ate //expectation → expectate

Zmena koncového „ator“ na „ate“: if (m()>1) ends(ator) → ate //operator → operate

Zmena koncového „alism“ na „al“: if (m()>1) ends(alism) → al

Zmena koncového „iveness“ na „ive“: if (m()>1) ends(iveness) → ive

//forgiveness→ forgive

Zmena koncového „fulness“ na „ful“: if (m()>1) ends(fulnes) → ful

//watchfulness → watchfull

Zmena koncového „ousness“ na „ous“: if (m()>1) ends(ousness) → ous //nervousness → nervous

Zmena koncového „ality“ na „al“: if (m()>1) ends(ality) → al //nationality → national

Zmena koncového „ivity“ na „ive“: if (m()>1) ends(ivity) → ive //activity → active

Zmena koncového „bility“ na „ble“: if (m()>1) ends(bility) → ble //inflexibility → inflexible

Zmena koncového „logi“ na „log“: if (m()>1) ends(logi) → log

Krok 4 – podobným spôsobom ako krok 3 upravuje koncovky -ic-, -full, -ness, atď.

Zmena koncového „icate“ na „ic“: certificate → bez zmeny
indicate → bez zmeny
ends(icate) → ic //publicate → public

Zmena koncového „ative“ na „ate“: representative → representate
ends(ative) → ate //cumulative → cumulate

Zmena koncového „alize“ na „al“: ends(alize) → al //libralize → liberal

Zmena koncového „icity“ na „ic“: ends(icity) → ic //publicity → public

Zmena koncového „ical“ na „ic“: radical → bez zmeny
ends(ical) → ic // political → politic

Odstránenie koncového „ful“: ends(ful) → „“ // unsuccessful → unsuccess

Odstránenie koncového „ness“: business → bez zmeny
ends(ness) → „“ // weakness → weak

Krok 5 – odstránenie -ant, -ence, atď.

Odstránenie koncového „ence“: experience → bez zmeny
if(m>0) ends(ence) → „“

Odstránenie koncového „ance“: circumstance → bez zmeny
ends(ance) → „“

Odstránenie koncového „er“: ends(ber) → bez zmeny //november, december, atď
minister, ever, liver, register, over, manager, officer,
consumer, manufacturer, newspaper, remainder, consider,
another, passenger → bez zmeny
if(m>0) ends(er) → „“

Odstránenie koncového „able“: if(m>0) ends(able) → „“ // available → avail

Odstránenie koncového „ible“: imposible → bez zmeny
if(m>0) ends(ible) → „“

Odstránenie koncového „ant“: significant → bez zmeny
if(m>0) ends(ant) → „“

Odstránenie koncového „ment“: unemployment → bez zmeny
if(m>0) ends(ment) → „“

Odstránenie koncového „ou“: if(m>0) ends(ou) → „“

Odstránenie koncového „ism“: mechanism, optimism, pesimism → bez zmeny
if(m>0) ends(ism) → „“

Odstránenie koncového „ous“: generous → bez zmeny
if(m>0) ends(ous) → „“
Odstránenie koncového „ive“: if(m>0) ends(ive) → „“
Odstránenie koncového „ize“: if(m>0) ends(ize) → „“

Krok 6 – najkomplikovanejší a najmenej elegantný krok. V tomto kroku sú ad-hoc pridávané koncovky -e, ktoré boli v predchádzajúcich krokoch odobraté navyše, napr.: requiring → requir → require, coming → com → come, atď. Tento krok bol kompletne prerobený, vzhľadom k tomu, že originálny algoritmus odstraňuje všetky koncovky „e“ bez rozdielu. Aplikovanie tohto kroku prináša rapídne zlepšenie korektnosti (správnosti) stemmingu.

Slovo končí na „c“:	ends(nc) → add(e)	//experiencing → experienc → experience; finance, france, counterbalance, advance, announce, pronounce
	ends(practic) → add(e)	//practicing → practic → practice
	ends(oduc) → add(e)	//introduced → introduc → introduce; produce
	ends(pric) → add(e)	//priced → pric → price
Slovo končí na „d“:	ends(lud) → add(e)	//exclude; include, conclude
	ends(trad) → add(e)	//trade
Slovo končí na „g“:	ends(ag) → add(e)	//encourage, average, envisage, leverage
	ends(verg) → add(e)	//diverge, converge
	ends(odg) → add(e)	//lodge
	ends(ang) → add(e)	//arrange, range, change
Slovo končí na „k“:	ends(mak) → add(e)	//make
	ends(tak) → add(e)	//take
	ends(fak) → add(e)	//fake
Slovo končí na „l“:	ends(coupl) → add(e)	//couple
	ends(handl) → add(e)	//handle
	ends(fil) → add(e)	//file
Slovo končí na „m“:	ends(com) → add(e)	//come
Slovo končí na „n“:	ends(clin) → add(e)	//decline
Slovo končí na „r“:	ends(uir) → add(e)	//acquire, require
	ends(nsur) → add(e)	//insure, ensure
	ends(mpar) → add(e)	//compare
	ends(structur) → add(e)	//restructure
	ends(assur) → add(e)	//assure
Slovo končí na „s“:	ends(chas) → add(e)	//purchase
	ends(hous) → add(e)	//house
	ends(pons) → add(e)	//response

ends(creas) → add(e) //increase

ends(rais) → add(e) //raise

ends(ris) & not(paris, chris, harris, paris)→ add(e) //authorise, arise, surprise, rise, comprise

ends(cis) & not(francis) → add(e) //exercise, criticise

ends(vis) → add(e) //revise, advise

ends(generalis) → add(e) //generalise

ends(fens) → add(e) //offensive

ends(aus) → add(e) //cause

ends(oppos) → add(e) //oppose

Slovo končí na „t“: ends(quot) → add(e) //quote

ends(ibut) → add(e) //attribute, distribute, contribute

ends(finit) → add(e) //definitive

ends(writ) → add(e) //underwrite

ends(complet) → add(e) //complete

Slovo končí na „u“: ends(issu) → add(e) //issue

Slovo končí na „v“: ends(ceiv) → add(e) //receive, conceive

ends(prov) → add(e) //improve, approve

ends(liev) → add(e) //relieve, believe

ends(solv) → add(e) //solve

ends(erv) → add(e) //serve, deserve, observe, reserve

Príloha 2: Ukážka slovníka nepotrebných slov

a	div	most	that	you
about	dlr	must	the	your
ad	do	new	their	
add	down	no	them	
after	dure	not	there	
all	each	now	they	
also	end	of	this	
an	expect	on	though	
and	five	one	three	
another	for	only	through	
any	four	or	to	
are	from	other	told	
around	go	out	two	
as	had	over	under	
ask	have	pct	until	
at	he	per	up	
be	his	plc	vs	
because	how	prior	was	
been	however	qtly	we	
before	i	reuter	week	
between	if	rev	were	
but	in	rmj	what	
by	inc	said	when	
can	into	share	where	
co	is	shr	which	
company	it	since	while	
continue	last	so	who	
could	like	some	will	
ct	may	stg	with	
cts	mln	such	without	
day	month	take	would	
did	more	than	year	

Príloha 3: Niektoré získané asociačné pravidlá z kolekcie textov Reuters-21578

asociačné pravidlo	**spoľahlivosť' (confidence)**
attack, iranian => gulf, monday, oil	90%
ship, u.s. => attack, iran, iranian, monday	88%
exchange, market, stock => price	67%
gain, profit => 1986, 1987, net	70%
debt, loan => bank	86%
opec, saudi => output, price, report	100
gerhard, stoltenberg => german, minister	100%
file, security => commission	100%
bank, debt => billion	71%
manage, manager, sell => fee, issue, lead, underwrite	100%
baker, james, secretary, trade => treasury, u.s.	88%
secretary, trade, u.s. => dollar	70%
japan, yen => bank, dollar	72%
1986, increase, level => rise	66%
offer, reject, twa => order, today, usair	100%
poor, salomon, standard => offer, point, price	80%
opec, price, well => official, oil, report	83%
president, treasury => baker, meet	93%
opec, production, quota => oil, price, report	100%
market, reform, subsidy => support, trade	100%
state, tax, toward => only, rate, reform	100%
trade, washington => country, international, market	66%
japan, problem => market, trade	77%
meet, monetary => fund, next	73%
country, international, monetary => bank	89%
currency, dollar => interest	66%
chairman, paul, term, volcker => alan, federal, greenspan, reserve	88%
reserve, u.s., volcker => alan, federal, greenspan	87%
tax, toward => market, reform, state	80%
louvre, treasury => german, interest, james, secretary, u.s., west	100%